15-110 Hw6 - Tweet Analytics

Hw6 and its check-ins are organized differently from the other assignments. If you haven't already done so, you should read the **Hw6 General Guide** to understand how this assignment works.

Project Description

Goal: To analyze a csv of tweets made by politicians using **pandas dataframes** for their sentiment. You will write functions to answer questions like *Which politician has the most attack tweets?* and *Which politician has the most negative tweets?* using the **nltk** library, and create visualizations using **matplotlib.** Part of the challenge of this assignment will be to look at documentation for these libraries yourself and figure out which functions will be most appropriate depending on the task.

Topics: natural language processing, data analysis, visualization, politics

Click on the following links to read the instructions for each week's assignment:

Hw6 Check-in 1 - due Monday 11/18

Hw6 Check-in 2 - due Monday 11/25

Hw6 - due Wednesday 12/04

Hw6 Check-in 1 - due Monday 11/18

In the first stage, you will organize the data for the project by installing necessary libraries and reformatting the CSV data into a pandas dataframe. This reformatting is necessary in order to perform analyses in the following stage.

Step 0: Written Assignment **[45pts]**

In addition to completing the steps described below, there is a short written assignment on the week's material. You can find the written assignment on Gradescope, and linked on the Assignments page of the course website.

Step 1: Installations **[0pts]**

The first goal of the first check-in is to make sure you are able to install the external libraries: pandas, nltk, matplotlib, and numpy. If you have installed them successfully, you should be able to run the starter code without any errors. After you have successfully installed all the libraries, you should proceed to Step 2!

This project will require a couple of installations for your personal machine. **Unfortunately, some of the required installations are not installed on the CMU cluster machines, so you need to own a laptop or desktop computer to complete this project.** We will hold sessions to help you install during the first week of the project, but you should be able to install the packages on your own by following these instructions.

To install the libraries you need, we recommend that you use the pip tool included in your Python installation. This tool will manage the installation process for you, which is much easier than trying to install a module manually. To use pip, open Terminal on a Mac/Linux or PowerShell on Windows. Then run the following lines of code:

pip install numpy
pip install pandas
pip install matplotlib
pip install nltk

If an error message occurs, try googling it to find a solution. TAs can also help debug installation errors via Piazza or in office hours.

Note that pip is associated with the default version of python on your computer. If you have multiple versions installed (which is often true of Macs, as they come with Python 2.7 installed by default), you will have to run a different command to install the libraries into the version of Python you use. These commands might work:

python3 -m pip install numpy
python3 -m pip install pandas
python3 -m pip install matplotlib
python3 -m pip install nltk

You can test whether the modules have installed into the correct version of Python by running the following commands in your interpreter. If they do not give you an error, you're good to go!

import numpy
import pandas
import matplotlib
import nltk

Note: some people have encountered errors with nltk not being able to read from URLs after it has installed. If you encounter an error along the lines of 'Error loading vader_lexicon: <urlopen error [SSL:', please follow these steps:

1. Find where Python is installed on your computer (you can do this by going to Pyzo > Shell > Edit Shell Configurations, and checking the location next to exe)
2. Go to that location on your computer (using Finder / File Explorer)
3. Double click the file 'Install Certificates.command' to run it

This should fix your problem! If these steps don't work, go to office hours for additional help.

Step 2: Generate Dataframes **[10pts]**

Currently, our tweet data is stored in a file type called csv, which stands for "comma separated values." This means that each value in each column is separated by a comma. You can check this out yourself by opening the file and seeing each row on a new line with each of its values separated by a comma!

In order to use this data, we want to convert the csv into a data structure that's easy to use with Python. Luckily, pandas is just the library for the job!

Given a filename of a .csv in the same folder as tweetanalytics.py, write a function makeDataFrame(filename) which takes in a string filename of a CSV file and returns a pandas dataframe. If you aren't sure how to do this, check the Data Analysis slides.

To test this function, run testMakeDataFrame().

Step 3: Parse Data **[15pts]**

Steps 3 and 4 are *helper functions,* and are grouped together for a more general purpose. The ultimate goal is to add 4 columns to our dataframe: Name, State, Position, and Region. In order to do so, we must parse this data out of the original tweet text.

First, write a function parseName(from_string) that takes a string of the form:
"From: <FirstName> <Lastname> (<Position> from <State>)"
and returns the name (first and last) in a single string.

For example:
parseName("From: Stephanie Rosenthal (Professor from Pennsylvania)") -> "Stephanie Rosenthal"

Next, write a function parsePosition(from_string) that takes a string of the form:
From: <FirstName> <Lastname> (<Position> from <State>)
and returns the position.

parsePosition("From: Stephanie Rosenthal (Professor from Pennsylvania)") -> "Professor"

Finally, write a function parseState(from_string) that takes a string of the form:
From: <FirstName> <Lastname> (<Position> from <State>)
and returns the state.

parseState("From: Stephanie Rosenthal (Professor from Pennsylvania)") -> "Pennsylvania"

To test these functions, run testParseName(), testParsePosition(), and testParseState().

Step 4: Find Region **[10pts]**

Write a function getRegionFromState(state_df, state) that takes a state_df dataframe

and a string of a state name to search for and return the corresponding region. The state_df has rows where each state is in the column 'State' and its corresponding region of the US, like Northeast or South, is in the 'Region' column.

Often, when looking up values in a dataframe, you know the value of one column, and you want to look up the associated value of a different column (like knowing a person's name and wanting to look up their phone number). To do this, use the code:

df.loc[df['known column name'] == 'known value to match', 'column name to return'].

Use this code to get the row associated with the state in the state dataframe using the correct column name and state value. Then, get the value of this cell in the dataframe by using row.values[0], and return this value.

To test this function, run testGetRegionFromState().

Step 5 Add Columns to Dataframe **[20pts]:**

Write a function addColumns(data, state_df) that takes in a dataframe data and a dataframe state_df, and destructively adds four new columns to data based on the data based on the functions you wrote above. Return None at the end of the function.

1. In order to add columns to a dataframe, you must make lists of all the values to add (one for each new column). Create four new empty lists for names, positions, states, and regions of the tweeters.
2. Uses iterrows() to iterate through each index and row in data. For each row, the function should:
    a. get the value in the column 'label' (the code looks the same as a dictionary with key 'label')
    b. call parseName, parsePosition, and parseState on that value.
    c. call getRegionFromState with state_df and the state you parsed to get its region.
    d. append each of the values to their respective list.
3. After the end of the for loop, set data['Name'], data['Position'], data['State'], and data['Region'] to the respective lists.
4. Return None

To test this function, run do_week1(). When the dataframe is printed at the end of the function, you should see some of your new columns at the end of the list of columns.

Hw6 Check-in 2 - due Monday 11/25

In this stage, you will dive deeper into actually analyzing the tweets in the CSV. You will use a popular natural language processing library called **nltk** to perform *sentiment analysis,* which will aim to predict how "positive" or "negative" a tweet is. You'll then analyze the most frequent bias & type of messages politicians tweet, and which states' politicians produce the most "attack" tweets. Finally, you will also analyze messages, tweets, and audiences by US Regions using the statemappings.csv from the first check-in.

Step 0: Written Assignment **[45pts]**

In addition to completing the steps described below, there is a short written assignment on the week's material. You can find the written assignment on Gradescope, and linked on the Assignments page of the course website.

Step 1: Identify Tweet Sentiment **[5pts]**

Edit the function findSentiment (classifier, tweet) to return "positive" if the classifier predicts the tweet is positive, "negative" if it is negative, and "neutral" if it is neutral. The first line of the code already runs a classifier on the tweet and receives a float score. You should add code to check if the score is less than -0.1 and if so return "negative"; if greater than 0.1, return "positive"; and otherwise return "neutral".

To test this function, run testFindSentiment().

Note: you might be curious about how the sentiment classifier works. Sentiment Analysis infers whether the statement has a positive, negative, or neutral connotation from the words in the sentence. This is a really hard problem. Think about the statement "That's great!" You could use it to mean it really is great, or you can use it in response to something that is really horrible. The sentiment classifier is trained using thousands of example sentences in many different contexts in order to help it understand whether particular words or phrases are generally positive, negative, neutral (or could be used both positively and negatively). This classifier generates a number for the text given to it, where negative numbers are associated with negative sentiment, and positive numbers are associated with positive sentiment.

Step 2: Add a Sentiment Column **[10pts]**

Edit the function addSentimentColumn(data) to perform a similar function to addColumns, but with sentiment.

Like last time, you should create a new empty list of sentiments. Then, use iterrows() to iterate through each index and row of the dataframe data. For each row, get the value of the 'text' column which represents the tweet, pass the tweet and the classifier to findSentiment, and append the result to the sentiments list. After the loop is finished, it should set data['Sentiment'] equal to the list of sentiments and return None.

To test this function, run testAddSentimentColumn() and check whether the returned list of values seems to have a sentiment in every row. You won't be able to see all the rows, but that's normal- there's a lot of data!

Step 3: Analyze Negative Sentiments **[10pts]**

For this rest of this stage, we will write functions that analyze the dataframe and create dictionaries mapping states or regions to data about that state or region.

First, write a function getNegSentimentByState(data) that creates a dictionary of tweet sentiment by the state of the tweeter. It should return a dictionary that maps each state to the number of messages from that state that had a negative sentiment.

To solve this problem, recall how we can generate dictionaries that map values to counts. Do the same thing here, but use row['Sentiment'] and row['State'] to access your values.

To test this function, run testGetNegSentimentByState().

Step 4: Analyze Attacks **[5pts]**

Write a function getAttacksByState(data) that creates a dictionary of number of tweet attacks by the state of the tweeter. This should use the same process as getNegSentimentByState(), except that you'll check whether the column 'message' is set to 'attack' instead of checking if 'Sentiment' is 'negative'.

To test this function, run testGetAttacksByState().

Step 5: Analyze Partisanship **[5pts]**

Write a function getPartisanByState(data) that creates a dictionary of number of partisan tweets by the state of the tweeter. This should use the same process as getNegSentimentByState(), except that you'll check whether the column 'bias' is set to 'partisan' instead of checking if 'Sentiment' is 'negative'.

To test this function, run testGetPartisanByState().

Step 6: Analyze Messages **[10pts]**

Write a function getMessagesByRegion(data) that returns a nested dictionary. The keys of the outer dictionary should be regions, which each map to an inner dictionary. The keys of the inner dictionary should be message types (like 'attack' from before). Each message type should map to the number of times it occurred in that region in the dataset.

To test this function, run testGetMessagesByRegion().

Step 7: Analyze Audiences **[10pts]**

Finally, write a function getAudienceByRegion(data) that returns a nested dictionary. The keys of the outer dictionary should again be regions, which each map to an inner dictionary. The keys of the inner dictionary should be audience types, from the column 'audience'. Each message type should map to the number of times it occurs in that region.

To test this function, run testGetAudienceByRegion()

Now that you've written all the analysis functions, you can run do_week2() and print the resulting dictionaries to check out your results!

Hw6 - due Wednesday 12/04

In the final part of this project, you will visualize all of the data you have collected in the previous check-ins. To do this, you will use matplotlib to create graphs and charts.

Step 0-A: Complete Check-in 1 **[20pts]**

If you got a perfect score on Hw6 Check-in 1 (the project part), congratulations; this step is already done! Go to the next step.

Otherwise, go back to your Gradescope feedback on Check-in 1 and use it to update your Check-in 1 code. This is your chance to implement any features you might have missed before, and fix any code that isn't working.

Step 0-B: Complete Check-in 2 **[20pts]**

If you got a perfect score on Hw6 Check-in 2 (the project part), congratulations; this step is already done! Go to the next step.

Otherwise, go back to your Gradescope feedback on Check-in 2 and use it to update your Check-in 2 code. This is your chance to implement any features you might have missed before, and fix any code that isn't working.

Step 1: Review Provided Code **[0pts]**

Drawing graphs with matplotlib requires a lot of setup code, and we want you to draw quite a few graphs, so we've provided a few functions for you to use, to simplify things. You will write some functions that call the functions we've provided.

You should definitely know what each of the following functions do, and we recommend that you look over their code at the bottom of the file quickly, to get a sense of how they work.

- **bar_plot:** generates a bar chart. The x values are the keys in the dictionary; the y values are the key's values.
- **sidebyside_bar_plots:** generates two bar charts side-by-side, for easy comparison. The x values are the values in the names list, and the y values for the two plots are the values in the values1 and values2 lists.

Step 2: Graph Attacks By States **[5pts]**

Write a function graphAttacksAllStates(dict) which takes a dictionary containing the counts of how many attack tweets were sent per state, and outputs a bar chart of the data. Use the provided function bar_plot, and don't forget to make a good title.

To test this function, check that the first graph produced by do_week3() has 37 bars, and that the largest bar is Texas.

Step 3: Graph Top N States **[15pts]:**

Write a function graphTopN(dict, n, title) which takes a dictionary including keys from all states and an n for the number of top states to select. It should create a new dictionary that includes only the top n states that have the highest counts in the original dictionary. Then create a bar chart of those states using bar_plot. Don't forget to include the provided title!

Hint: it may be useful to loop through the original dictionary n times, each time finding the top state that isn't in the new dictionary already.

To test this function, check the second and third graphs produced by do_week3(). The second should have five bars: Texas, California, Louisiana, Florida, and Pennsylvania. The third should have five bars: Texas, California, Florida, North Carolina, and New York

Step 4: Graph a Region Comparison **[20pts]:**

Write a function graph2Regions(dict, r1, r2, title) which takes a dictionary of dictionaries and two regions r1 and r2 (which are keys in the dictionary), finds the dictionary values of those regions, and graphs those values in a side-by-side bar chart. In order to do this, you will need to make lists out of the dictionaries.

Make a list of all the keys in r1 and r2, with no duplicates. Then make two lists: one of the dictionary values in r1, one of the values in r2 (or 0 if a key doesn't appear). Each value at index i of the two lists should correspond to the key at index i of the original list.

Finally, use sidebyside_bar_plots(names, values1, values2, category1, category2, title) with the key list as names, the two lists of values, as values1 and values2, the category names (r1 and r2), and the given title, to make the graph.

To test this function, check the fourth and fifth graphs produced by do_week3(). The fourth should have 9 bar-comparisons, with orange (South) as the highest value on each comparison. The fifth should have two comparisons, again with orange (South) as the highest value on both.

Step 5: Compare Sentiment to Attack Counts **[20pts]:**

Finally, write a function graphSentCountAttackCount(sentiments, attacks, title) which gets two dictionaries, one mapping states to negative sentiment counts, the other mapping states to attack counts, and creates two lists - one for the values of negative sentiment counts and one for values of attack counts for each corresponding state at the same index. Append 0 if the state is not in the dictionary. Then, you should use sidebyside_bar_plot to graph the corresponding values, similarly to how you created the graph in the previous function.

To test this function, check the sixth graph provided by do_week3(). This should have 49 comparisons overall.

Congratulations- now you're done! Enjoy looking through the different graphs you can generate to investigate the sentiments of different political tweets.