CSCI 140 Project 3 Part 1

Due Friday, October 9 2020 by 1700 (5:00 PM EST)

For the third project, you will be creating a suite of functions to process and analyze Twitter data. You will use your functions to analyze a Twitter data set obtained via the Twitter API from August 9 through August 31. The data was downloaded by searching for tweets with two keywords: tortoise and umbrella and by excluding re-tweets (see below for a definition). This data was approved by Twitter for educational use. **Please be aware that this is real, raw data downloaded from the actual Twitter platform and thus contains information that is not under the control of the course instructors. If you discover content you find objectionable, please alert the instructors.** Web links have been removed from the tweets in order to reduce the amount of unfiltered content.

For Part 1 of this project, you will write 2 functions and correct 2 functions that you will use in a short main program to process and analyze the data. You are provided with three text files, as well two Python skeleton files (Project_3.py and Project_3_Main.py). You will submit your functions in Project_3.py and your main program in Project_3_Main.py.

BE AWARE: You are not permitted to use NLTK. We will use NLTK for Project 3 Part 2.

About Twitter data and Tweets

Tweets are submission to the social media platform Twitter. They are 280 characters in length or less. Tweets often contain hashtags which are words or phrases with the prefix #, for example, #avocadotoast. Tweets may reference other Twitter users, as indicated by the @, for example, @williamandmary. Users may re-post a tweet posted by another user – this is referred to as re-tweeting, and is signified by RT followed by the user who originally posted it, for example, RT @POTUS. The tweets you will be examining for the project will contain links which have the prefix https or http. They will also contain emojis, for example: 🤔.

Some of the tweets in this project obtained via the Twitter API are truncated, i.e., the whole tweet is not shown. These tweets will have an ellipsis (an ellipsis is ...) at then end of the text and may or may not have a link after, for example:

I'm sad you are leaving. I thought you were the wisest, clearest and most honest voice on the Comm…

We will not worry about this for this part of the project.

Setup for Working with Emojis

These instructions are a duplicate of those on the Module 3 main page. If you have already installed emoji, you do not need to do it again. Part of this project entails the characterization of emojis in tweets. In order to do this you will need to install the emoji package. Open a terminal and run this command: `pip install emoji`
Enter Y at any prompts that come up.

Testing your Code

The best way to test your code is NOT to use the large files with many tweets. Make a small text file that you can easily count the number of hashtags, emojis, and mentions in. Use that file for testing. Better....make multiple small files for testing! You have been given one file to use for testing: **test_file.txt** but please be aware that it does not contain every possible case.

Part 1: Extracting features

All of the code for Part 1 should be submitted in the Project_3.py file. You will fill in your functions under the definition lines.

Task 1: *process_text(text, punct = '.,;:"?!.')*

One of our goals will be to look at the usage of hashtags and user mentions. It is useful to pre-process text to eliminate things that may make us miss individual hashtags (e.g. the lack of a space between the two hashtags in #turtle#tortoise) and/or count hashtags as being different when they really aren't (e.g. #turtle vs. #turtle???)

`process_text` takes one required input, a string, **text**. The function also takes an optional argument **punct** which is a string of punctuation symbols to remove. This function reformats the text to make it ready for further processing as follows:

- Since we are interested in hashtags and mentions, we will insert a space in front of every #, and in front of every @. This is not as difficult as it sounds, think of it as turning a # into a # with a space in front of it, and the same for @.
- It will also deal with punctuation. Replace dashes (hyphens -) with a space. Delete whatever punctuation is specified by the optional argument **punct**.
- The function returns a **list** that is the remaining words from the processed tweet. The function should print nothing.

Here are some examples. This is not actual code, and words like "Input text" should not print when your function runs. These examples show text that is a single tweet, but keep in mind that this function could be used on text of any length, including a whole file of tweets.

Input text: **Taco trucks on every corner? Business group wants them at every polling site instead.#politics#tacos**

Default value of **punct**

process_text returns: `['Taco', 'trucks', 'on', 'every', 'corner', 'Business', 'group', 'wants', 'them', 'at', 'every', 'polling', 'site', 'instead', '#politics', '#tacos']`

Input text: **'Sen. Sanders proposes one-time tax that would cost Bezos $42.8 billion, Musk $27.5 billion'**

punct is '.;$'

process_text returns: `['Sen', 'Sanders', 'proposes', 'one', 'time', 'tax', 'that', 'would', 'cost', 'Bezos', '428', 'billion,', 'Musk', '275', 'billion']`

Key points:

- The input **text** is a string. If you are reading in from a file, you are doing it wrong.
- **text** is a string, but in the course of the function running, it should become a list.
- Notice that we are NOT changing the case of the text. We are considering case differences as unique hashtags and mentions (so @userNAME is different from @username)
- For this project, you are not permitted to use NLTK. We will do that in the next project. Complete this function using string/list methods and other structures you already know.
- There are different characters for hyphens – sometimes the character used in Windows does not match up with what is in text files. It's OK as long as your code is correct.
- The #1 question that gets asked with functions like this is: why does my list print every word on a separate line? It's because you are in Jupyter notebook and didn't actually use print....just the notebook's Out function. Put your function call inside print.

Task 2: *get_emojis(text)*

The function **get_emojis** takes a single input **text** which is a string. It returns a list containing any emojis found in **text**. Here are a few examples.

text is: '@wonderofscience Shoutout to elon musk 😰'

get_emojis returns: ['😰']

text is: '#Turtles #Tortoise Rainbow 🌈 the Therapy Tortoise🐢 We took Rainbow the leopard tortoise to visit granny today.

get_emojis returns: ['🌈', '🐢']

A few things to consider:

- Some text may have no emojis – in this case, the function returns an empty list.
- Some text may have the same emoji repeated several times – in this case the function returns a list containing every occurrence, e.g., if '🐢' appears three times, the list will have ['🐢', '🐢', '🐢'].
- There may not necessarily be a space between an emoji and the word before or after it. For example, notice that in: 'the Therapy Tortoise🐢' – there is no space between tortoise and the emoji

The code you have been given for **get_emojis** is incorrect and needs to be debugged. You are not permitted to add lines and must leave the code as list comprehension. The changes are relatively minor. If you are looking up modules and structures we have not learned (lambdas, generators, itertools) to try and use, you are doing it wrong. If you are trying to add a second for loop, you are doing it wrong.

You may find it helpful to expand the code given, then write the code WITHOUT list comprehension to identify the errors and then turn it back into list comprehension at the end.

Task 3: *get_features(text_list, feature)*

The function **get_features** takes two required inputs: **text_list** is a list and **feature** is a string consisting of a single character. **feature** will always be either `'#'` or `'@'` - you can assume that the user will never input anything besides one of those two characters. **get_features** returns a list containing all of the hashtags in **text_list** when **feature** is `'#'`; it returns a list containing all of the user mentions in **text_list** when **feature** is `'@'`. Here are a few examples:

text_list is: ['@wonderofscience', 'Shoutout', 'to', 'elon', 'musk', '😰']

feature is `'@'`

get_features returns: ['@wonderofscience']

text_list is: '['#Turtles', '#Tortoise', 'Rainbow', '🌈', 'the', 'Therapy', 'Tortoise🐢', 'We', 'took', 'Rainbow', 'the', 'leopard', 'tortoise', 'to', 'visit', 'granny', 'today.']

feature is `'#'`

get_features returns: [`'#Turtles', '#Tortoise'`]

A few things to consider:

- The input is a **list** not a string
- Some inputs may have no features– in this case, the function returns an empty list
- Some inputs may have the same hashtag or user mention repeated several times – in this case the function returns a list containing every occurrence
- This function only processes mentions or hashtags, it is not finding both at the same time. So the list returned has only hashtags or only mentions – this is determined by the argument **feature** as shown above
- Notice that **get_features** is not responsible for changing the case of items or processing punctuation. Your function should not do any of those things.

The code you have been given for **get_features** is incorrect and needs to be debugged. You are not permitted to add lines and must leave the code as list comprehension. The changes are relatively minor. If you are looking up modules and structures we have not learned (lambdas, generators, itertools) to try and use, you are doing it wrong. If you are trying to add a second for loop, you are doing it wrong.

You may find it helpful to write the code WITHOUT list comprehension to identify the errors and then turn it back into list comprehension.

Task 4: *feature_extract(filename, feature = '#')*

Write a function called **feature_extract** which returns a dictionary.

The required argument **filename** is a string that corresponds to the name of a file containing tweets. The required argument **feature** will always be one of these three strings: `'#'` , `'@'`, or `'emoji'` – you can assume that the user always inputs one of these three.

The dictionary returned by **feature_extract** will contain strings as keys and integer counts as values. The keys will be hashtags (see explanation on page 1) if **feature** is `'#'`**,** user mentions (see explanation on page 1) if **feature** is `'@'` or emojis if **feature** is `'emoji'`. The values will be the number of occurrences of each individual hashtag, mention, or emoji.

Within this function, you must make use of your three other functions (process_text, get_features, get_emojis) to receive full credit. Why? Using your other functions will make this easier for you.

The examples below all assume the same input file. Please note that each example is a separate execution of the function (i.e. this represents 3 separate function calls).

Input file contains:

#Turtles #Tortoise Tootles, Otis and Clover via /r/turtles By:OwOviaa - https://t.co/hWYuFmMfeO
#Turtles #Tortoise Torti-lla?! Happy Saturday everyone. Make it a Fiesta via /r/tortoise
#Turtles #Tortoise Rainbow 🌈 the Therapy Tortoise 🐢 We took Rainbow the leopard tortoise to visit granny today.
How do you feel today, tortoise or hare? 😹💪🔝 #newtron#retrofit#newtrontechnology #fullelectric #electricvehicle
@gtconway3d Mark must must be the tortoise of readers 😹😹😹

feature is `'#'`
feature_extract returns: `{'#Turtles': 3, '#Tortoise': 3, '#newtron': 1, '#retrofit': 1, '#newtrontechnology': 1, '#fullelectric': 1, '#electricvehicle': 1}`

feature is `'@'`
feature_extract returns: `{'@gtconway3d': 1}`

feature is `'emoji'`
feature_extract returns: `{'🌈': 1, '🐢': 1, '😹': 4, '💪': 1, '🔝': 1}`

Things to consider:

- Go back over the specific details for **get_emoji** and **get_features** – they give you some hints with respect to processing, and possible pitfalls in the way the text may be written.
- This function returns a dictionary. Make sure you are returning a dictionary, not a list.
- Think about how to optimize your code. It sounds like there are 3 cases here.....but could you write this function using only 2 cases? You can expect to be evaluated on the efficiency of your code.
- Remember, you MUST use your other functions in this function for full credit.
- Some emojis may not print as emojis, they may appear in your dictionary as a text string, like this: \U0001f97a – that is OK. It's related to your operating system and text encoding. This is not an error in your code and does not affect grading.
- Create the dictionary using dictionary comprehension and earn some extra credit!

Part Two: Main Program

For this part of the assignment you will use your functions to process two Twitter data sets, stored in two separate files. These are the Tortoise data set (tortoise_tweets.txt) or the Umbrella data set (umbrella_tweets.txt).

- Create a dictionary containing all of the emojis from the Tortoise data set. Be sure to assign a variable name to this dictionary.
- Create a dictionary containing all of the emojis from the Umbrella data set. Be sure to assign a variable name to the dictionary.
- Find all of the emojis that are shared by the two data sets (i.e. the emojis that can be found in both) and store them in an object – print out the object you've stored in them so we can see them. There are many ways to do this, but think simple. (Hint: what data structure did we learn about, but haven't actually used in this project yet?)
- Find all of the emojis that are in the Tortoise data set but not the Umbrella data set and store them in an object – print out the object you've stored in them so we can see them.. There are many ways to do this, but think simple. (Hint: what data structure did we learn about, but haven't actually used in this project yet?)
- Create a dictionary containing all of the hashtags from the Umbrella data set. Print this dictionary out. What is the most common hashtag? (You don't have to write code to find it, just look at the data, and comment on it in your write-up.)

SUBMISSION EXPECTATIONS

`Project_3.py` Your implementations/correction of the functions in Part 1.

`Project_3_Main.py` Your main program for Part 3. The first line in this file should be:

```
from Project_3 import *
```

`Project_3_Part_1.pdf` A PDF document containing your reflections on the project. **You must also cite any sources you use. Please be aware that you can consult sources, but all code written must be your own. Programs copied in part or wholesale from the web or other sources will result in reporting of an Honor Code violation.**

POINT VALUES AND GRADING RUBRIC

This part of the project is worth 17.5 points and Part 2 of the project is worth 7.5 points.

Part 1:

- process_text (3 pts)
- get_emojis correction (2.5 pts)
- get_features correction (2.5 pts)
- feature_extract (6 pts)

Part 2: Main Program (3.25 pts)

Writeup – 0.25 pts

Grade Level	Execution	Correctness and Algorithms	Formatting	Style and References
A	Code executes without errors, requires no manual intervention	Code correctly implements all algorithms, passes all test cases (with possible exception of extremely rare case as applicable), and meets all required functionality	Inputs and outputs formatted as per the specification both in terms of content and type, may be minimal formatting issues (spacing, punctuation, etc.); correct file formats and names	Code uses current structures and modules discussed in class; meaningful variable names; commented as appropriate; any references cited in write-up
B	Code executes without errors, may require minimal manual intervention such aschange of an import line	Meets all required functionality with exception of minor/ rare test cases (as applicable)	Inputs and outputs formatted as per the specification both in terms of content and type, some minor formatting issues may be present (e.g. incorrect string text), correct file formats and names	Code uses current structures and modules discussed in class; meaningful variable names; commented as appropriate; any references cited in write-up
C	Code does not execute without minor manual intervention such as correcting indentation or terminating parentheses or a single syntax error	Code contains logical errors and fails subset of both rare and common test cases, overall algorithmic structure reflects required functionality but implementation does not meet standards	Inputs and outputs have major formatting issues, files incorrectly named but file formats correct	Code uses current structures and modules discussed in class; meaningful variable names; commented as appropriate; references cited in write-up
D	Code does not execute without major manual intervention such as changing variable names, correcting multiple syntax errors, algorithmic changes	Code contains major logical errors and fails majority of test cases, lack of cohesive algorithmic structure, minimal functionality	Inputs and outputs have major formatting issues, files incorrectly named but file formats correct	Code uses some structures and modules from class, but otherwise largely taken from outside sources with citations; variable names and code structure hard to decipher
F	Code requires extensive manual intervention to execute	Code fails all or nearly all test cases and has no functionality to solve the proposed problem	Inputs and outputs have major formatting issues, files in incorrect formats which cannot be graded	Code uses structures and modules from outside sources only with no citations; variable names and code structure hard to decipher; code written in other programming languages or Python 2.7