Twitter Sentiment Analysis in Python: Instructions

(Adapted from Bill Howe's programming assignment, Coursera course Introduction to Data Science)

In this assignment, you will

- access the Twitter Application Programming Interface(API) using python
- estimate the public's perception (the *sentiment*) of a particular term or phrase
- calculate the frequency of terms in a twitter file

Some points to keep in mind:

- This assignment is open-ended in several ways. You'll need to make some decisions about how best to solve the problem and implement them carefully.
- **It is perfectly acceptable to discuss your solution on the forum, but don't share code.**
- **Each student must submit his/her own solution to the problem.** The code will be submitted to plagiarism detection software (Moss) to detect if students are submitting the same code.
- You will submit your code, and the professor or grader will run your code against a test data set to see if your code produces the correct answer.
- Your code should only use the Python standard libraries unless you are specifically instructed otherwise. Your code should also not rely on any external libraries or web services.

What you will turn in:

- A file called `lastname_firstname_assignment1.zip` that contains three files:
    - A text file containing first 20 lines of downloaded twitter data called: `lastname_firstname_first20.txt`
    - Program to derive sentiment of each English tweet in downloaded twitter data called: `lastname_firstname_tweet_sentiment.py`
    - Program to calculate frequency of term occurrences in downloaded twitter data called: `lastname_firstname_frequency.py`

The Twitter Application Programming Interface

Twitter provides a very rich REST API for querying the system, accessing data, and control your account. You can read more about the Twitter API

Python environment

If you are new to Python, you may find it valuable to work through the codeacademy Python tutorials. Focus on tutorials 1-9, plus tutorial 12 on File IO. In addition, many students have recommended Google's Python class.

You will need to establish a Python programming environment to complete this assignment. You can install Python yourself by downloading it from the Python website.

Unicode strings

Strings in the twitter data prefixed with the letter "u" are unicode strings. For example:

```
u"This is a string"
```

Unicode is a standard for representing a much larger variety of characters beyond the roman alphabet (greek, russian, mathematical symbols, logograms from non-phonetic writing systems such as kanji, etc.)

In most circumstances, you will be able to use a unicode object just like a string.

If you encounter an error involving printing unicode, you can use the encode method to properly print the international characters, like this:

```
unicode_string = u"aaaÃ Ã§Ã§Ã§Ã±Ã±Ã±"

encoded_string = unicode_string.encode('utf-8')

print encoded_string
```

Problem 1: Get Twitter Data

To access the live stream, you will need to install the **oauth2 library** so you can properly authenticate. You can install it yourself in your Python environment. (The command `$ pip install oauth2` should work for most environments, or try easy-install oauth2.)

The steps below will help you set up your twitter account to be able to access the live 1% stream.

1. Create a twitter account if you do not already have one. Respond to the resulting email from Twitter to Confirm your Twitter Account.
2. Go to **https://dev.twitter.com/apps** and log in with your twitter credentials.
3. Click "Create New App"
4. Fill out the form and agree to the terms. Put in a dummy website if you don't have one you want to use.
5. On the next page, click the "API Keys" tab along the top, then scroll all the way down until you see the section "Your Access Token"
6. Click the button "Create My Access Token". You can **Read more about Oauth authorization.**
7. You will now copy four values into twitterstream.py. These values are your "API Key", your "API secret", your "Access token" and your "Access token secret". All four should now be visible on the API Keys page. (You may see "API Key" referred to as "Consumer key" in some places in the code or on the web; they are synonyms.) Open twitterstream.py and set the variables corresponding to the api key, api secret, access token, and access secret. You will see code like the below:

   ```
   api_key = "<Enter api key>"
   api_secret = "<Enter api secret>"
   access_token_key = "<Enter your access token key here>"
   access_token_secret = "<Enter your access token secret here>"
   ```

8. Run the following and make sure you see data flowing and that no errors occur.

   ```
   $ python twitterstream.py > output.txt
   ```

   This command pipes the output to a file. Stop the program with Ctrl-C, but wait at least **3 minutes** for data to accumulate. Keep the file output.txt for the duration of the assignment; we will be reusing it in later problems. Don't use someone else's file; we will check for uniqueness in other parts of the assignment.

What to turn in: The first 20 lines of the twitter data you downloaded from the web. You should save the first 20 lines to a file `lastname_firstname_first20.txt` by using the following command:

```
$ head -n 20 output.txt > lastname_firstname_first20.txt
```

Problem 2: Derive the sentiment of each tweet

For this part, you will compute the sentiment of each tweet based on the sentiment scores of the terms in the tweet. The sentiment of a tweet is equivalent to the sum of the sentiment scores for each term in the tweet.

You are provided with a skeleton file `tweet_sentiment.py` which accepts two arguments on the command line: a *sentiment file* and a tweet file like the one you generated in Problem 1.

Rename your file to be `lastname_firstname_tweet_sentiment.py`

You can run the skeleton program like this:

```
$ python lastname_firstname_tweet_sentiment.py AFINN-111.txt ouptput.txt
```

The file AFINN-111.txt contains a list of pre-computed sentiment scores. Each line in the file contains a word or phrase followed by a sentiment score. Each word or phrase that is found in a tweet but not found in AFINN-111.txt should be given a sentiment score of 0. See the file AFINN-README.txt for more information.

Note that all the terms in AFINN-111.txt are lower case. You may want to use the function `mystring.lower()` to convert your tweet to all lower case for this problem.

To use the data in the AFINN-111.txt file, you may find it useful to **build a dictionary**. Note that the AFINN-111.txt file format is tab-delimited, meaning that the term and the score are separated by a tab character. A tab character can be identified a "\t". The following snippet may be useful:

```
afinnfile = open("AFINN-111.txt")
scores = {} # initialize an empty dictionary
for line in afinnfile:
  term, score  = line.split("\t")  # The file is tab-delimited.
       # "\t" means "tab character"
  scores[term] = int(score)  # Convert the score to an integer.

print scores.items() # Print every (term, score) pair in the dictionary
```

The data in the tweet file you generated in Problem 1 is represented as *JSON*, which stands for JavaScript Object Notation. It is a simple format for representing nested structures of data --- lists of lists of dictionaries of lists of .... you get the idea.

Each line of `lastname_firstname_first20.txt` represents a streaming message. Most, but not all, will be tweets. (The skeleton program will tell you how many lines are in the file.)

It is straightforward to convert a JSON string into a Python data structure; there is a library to do so called `json`.

To use this library, add the following to the top of `tweet_sentiment.py`

```
import json
```

Then, to parse the data in `output.txt`, you want to apply the function `json.loads` to every line in the file.

This function will parse the json data and return a python data stucture; in this case, it returns a dictionary. If needed, take a moment to **read the documentation for Python dictionaries**.

You can read the Twitter documentation to understand what information each tweet contains and how to access it, but it's not too difficult to deduce the structure by direct inspection.

Your script should print to stdout the sentiment of each tweet in the file, one numeric sentiment score per line. The first score should correspond to the first tweet, the second score should correspond to the second tweet, and so on. If you sort the scores, they won't match up. If you sort the tweets, they won't match up. If you put the tweets into a dictionary, the order will not be preserved.

The nth line produced by the file you submit should contain only a single number that represents the score of the nth tweet in the input file!

NOTE: You must provide a score for **every** tweet in the sample file, even if that score is zero. You can assume the sample file will only include English tweets and no other types of streaming messages.

This is real-world data, and it can be messy! Refer to the twitter documentation to understand more about the data structure you are working with. Don't get discouraged, and ask for help on the forums if you get stuck.

Your code should check whether each line read from the output.txt files is a valid English tweet; if not, set its sentiment score to zero.

The file `AFINN-111.txt` contains all lower case terms, so be sure to use the `mystring.lower()` function to convert your string to lower case before checking the sentiment of each term.

To remove ASCII punctuation from your string, first import the string library at the start of your program:

```
import string
```

Then use the translate function to remove ASCII punctuation from your Unicode string:

```
newstring = mystring.encode('utf-8').translate(None, string.punctuation)
```

Note that this only removes ASCII punctuation, so some Unicode punctuation may remain.

To grade your submission, we will run your program on a tweet file formatted the same way as the `output.txt` file you generated in Problem 1.

You may test your program on a short tweet file provided called `shortTwitter.txt` and compare your output to the solution file `tweet_sentiment_shortTwitter_solution.txt`.

What to turn in: `lastname_firstname_tweet_sentiment.py` after you've verified that it returns the correct answers. You can test your code using the test.txt input file and check the results against the tweet_sentiment_solution file.

Problem 3: Compute Term Frequency

Write a Python script `lastname_firstname_frequency.py` to compute the *term frequency histogram* of the livestream data you harvested from Problem 1.

The frequency of a term can be calculated as `[# of occurrences of the term in all tweets]/[# of occurrences of all terms in all tweets]`

Your script will be run from the command line like this:

```
$ python lastname_firstname_frequency.py output.txt
```

You should assume the tweet file contains data formatted the same way as the livestream data.

You may again want to make use of a dictionary to associate each term that occurs in a tweet with the number of occurrences of that term.

Your script should print output to stdout. Each line of output should contain a term, followed by a space, followed by the frequency of that term in the **entire file**. There should be one line per **unique** term in the entire file. Even if 25 tweets contain the word `lol`, the term `lol` should only appear once in your output. Each line should be in the format `<term:string> <frequency:float>`

For example, if you have the pair `(bar, 0.1245)` in Python it should appear in the output as:

```
bar 0.1245
```

If you wish, you may consider a term to be a multi-word phrase, but this is not required. You may compute the frequencies of individual tokens only.

Depending on your method of parsing, you may end up computing frequencies for hashtags, links, stop words, phrases, etc. If you choose to filter out these non-words, that's ok too.

You may test your program on a short tweet file provided called `shortTwitter.txt` and compare your output to the solution file `frequency_shortTwitter_solution.txt`.

What to turn in: The file `lastname_firstname_frequency.py`