Programming Assignment 1: Sentiment Analysis of Twitter Data

Twitter has emerged as a fundamentally new instrument to obtain social measurements. For example, researchers have shown that the "mood" of communication on twitter can be used to predict the stock market.

In this programming assignment you will:

- Load and prepare a collected set of twitter data for analysis
- You will estimate the sentiment associated with individual tweets
- You will estimate the sentiment of a particular term

Please keep in mind the following points:

- This assignment is open-ended in several ways. You will need to make some decisions about how to best solve each of the problems mentioned above.
- **It is absolutely fine to discuss your solutions with your classmates (either in person or in Piazza) but you are not allowed to share code.**
- **Each student must submit their own solution via CANVAS.**
- You will have unlimited number of uploads to CANVAS.

Formatting of Twitter Data

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 mach larger variety of characters beyond the roma alphabet (greek, russian, mathematical symbols, logograms from non-phonetic writing systems, 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 (https://docs.python.org/3/library/stdtypes.html#str.encode) method to properly print the international characters. You can find more information about UNICODE and Python 3 here (https://docs.python.org/3/howto/unicode.html).

Question 1: Loading and Cleaning Twitter Data [20 points]

In this first part, you will neeed to load a sample of tweets in memory and prepare them for analysis. The tweets are stored in the file `tweets.json`. This file follows the *JSON* format. JSON stands for JavaScript Object Notation. It is a simple format for representing nested structres of data --- lists of lists of dictionaries of lists of ... you get the idea.

Each line in of `tweets.json` represents a message. It is straightforward to convert a JSON string into a Python data structure; there is a library to do so called `json`. Below we will show you how to load the data and how to parse the first line in the `tweets.json` file.

In [ ]:
```
import json

# Open the input file
input_file = open('tweets.json','r')


# Load the first few lines
line_count = 10
for line in input_file:
    tweet = json.loads(line)
    print(tweet)
    line_count -= 1
    if line_count < 0:
        break


# Close the input file
input_file.close()
```

Each entry in `tweets.json`, i.e., each `tweet`, corresponds to a dictionary that contains lots of information about the tweet, the user, the activity related to the tweet (i.e., if it was retweeted or not), the timestamp of the tweet, entities mentioned in the tweet, hashtags used, etc.

You can treat the `tweet` variable from above as a dicitonary and use the `.keys()` command to see the fields associated with the dictionary.

In [ ]:
```
for k in tweet.keys():
    print(k)
```

We can select any of the aforemented values of Variable `tweet` by treating it as a dictionary. For example let's select the `text` body of the tweet, the time it was `created_at`, and the `hashtags` it contains.

In [ ]:
```
body = tweet['text']
tweet_time  = tweet['created_at']

# tweet['entities'] is itself a dictionary with more entries. Hashtags
is one of them.
htags  = tweet['entities']['hashtags']

print('Tweet body: ', body)
print('Creation Time:', tweet_time)
print('Hastags: ', htags)
```

As you can see this tweet contains no hashtags. The body of the tweet contains several information that is not necesary for our sentiment analysis task. For example, it contains a comma, a reference to a twitter user and a link to an external website.

Since this information is not necessary we can remove it. In other words we need to clean our input in order to prepare it for analysis. Next, we show you some basic cleaning operations using **regular expressions**. You can find more information on regular expressions here (https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285).

In [ ]:
```
# Basic steps for cleaning process.
import re
# Step 1: Convert tweet to lower case
body = body.lower()
# Step 2: Find URLs and replace them with an empty string
body = re.sub(r'((www\.[\S]+)|(https?://[\S]+))', '', body)
# Step 3: Find @<user> mentions and replace them with an empty string
body = re.sub(r'@[\S]+', '', body)

print('Clean tweet body:', body)
```

We are providing you with a Python script named `preprocess.py`. The script `preprocess.py` accepts one argument on the command line: a JSON file with tweets (i.e., `tweets.json`). You can run the program like this:

```
$ python3 preprocess.py tweets.json
```

There are some parts specified in this script that you need to implement. The goal of this script is to clean all the tweets in `tweets.json`. Running `preprocess.py` will generate an output file named `clean_tweets.txt` containing **one string per line** containing a clean tweet. The order of the clean tweets in your output file should follow the order of the lines in the original `tweets.json`. Basically, the first line in `clean_tweets.txt` should correspond to the first raw tweet in `tweets.json`, the second line should correspond to the second tweet, and so on. If you perform any sorting or you put the processed data in a dictionary the order will not be preserved. Once again: **The n-th line of `clean_tweets.txt` (the file you will submit) should be a string that represent the clean version of the n-the line in the `tweets.json` (the input file).**

You must provide a line for **every** tweet. If the clean tweet is the empty string then just provide a line with the empty string.

If you have implemented everything correctly, the first 6 lines of the generated output should be exactly the same as the next lines:

```


any shots
a cancer may act shy and quiet but will adamantly defend a loved one again
st outsiders
```

The first 4 lines will be empty and the last 2 will have the string shown above.

Note: This is real-world data, and it can be messy! For example, not all json lines may contain valid tweets (i.e., a text field). Ask for help on Piazza if you get stuck!

What to turn in: The file `clean_tweets.txt` output by `preprocess.py` after you have implemented the missing parts in `preprocess.py`.

Question 2: Derive the sentiment of each tweet [40 points]

For this part, you will compute the sentiment of each clean tweet in `clean_tweets.txt` 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 clean 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 Question 1. You can run the skeleton program like this:

```
$ python3 tweet_sentiment.py AFINN-111.txt clean_tweets.txt
```

The file `AFINN-111.txt` contains a list of pre-computed sentiment scores. Each line in the file contains a word or phrase phollowed 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.

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 corresponds to the string "\t". The following snipped of code may be useful:

In [ ]:
```
import sys
afinnfile_name = open(sys.argv[1])
afinnfile = open(afinnfile_name, 'r')
scores = {} # initialize an empty dictionary
for line in afinnfile:
    term, score = line.split("\t") # The file is tab-delimited and "\t
" means tab character
    scores[term] = int(score) # Conver the score to an integer. It was
parsed as a string.
afinnfile.close()
print(scores.items( )) # Print every (term, score) pair in the diction
ary
```

Your script should output a file named `sentiment.txt` containing the sentiment of each tweet in the file `clean_tweets.txt`, 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. In other words, **the n-th line of the file you submit should contain only a single number that represents teh score of the n-th tweet in the input file.**

After you have implemented everything the first 10 lines of the generated output of your script should be exactly the same as the next lines:

```
0
0
0
0
0
6
6
-10
0
0
```

What to turn in: The file `sentiment.txt` after you have verified that it returns the correct answers

Question 3: Derive the sentiment of new terms [40 points]

In this part you will create a script that computes the sentiment for terms that **do not** appear in the file `AFINN-111.txt`.

You can think about this problem as follows: We know wwe can use the sentiment-carrying words in `AFINN-111.txt` to deduce the overall sentiment of a tweet. Once you deduce the sentiment of a tweet, you can work backwards to deduce the sentiment of the non-sentiment carrying words that *do not appear* in `AFINN-111.txt`. For example, if the word *football* always appears in proximity with positive words like *great* and *fun*, then we can deduce that the term *football* itself carried a positive sentiment.

You are provided with a skeleton file `term_sentiment.py` which accepts the same two arguments as `tweet_sentiment.py` and can be executed using the following command:

```
$ python3 term_sentiment.py AFINN-111.txt clean_tweets.txt
```

Your script should print its output to stdout. Each line of the output should contain a term, followed by a space, followed by a sentiment. That is, each line should be in the format term:string (term:string) sentiment:float (sentiment:float). For example if you have the pair ("foo", 54.2) in Python, it should appear in the output as: `foo 54.2`.

The order of your output does not matter.

What to turn in: The file `term_sentiment.py` after you have implemented the missing parts.

To grade your submission we will run your script on a file that contains strongly positive and strong negative tweets and verify that the terms in the strongly positive tweets are assigned a higher score than the terms in the negative tweets. Your score need not exactly match any specific solution.