Twitter Sentiment Analysis in Python: Instructions

When you're ready to submit your solution, go to the assignments list.

Twitter represents a fundamentally new instrument to make social measurements. Millions of people voluntarily express opinions across any topic imaginable --- this data source is incredibly valuable for both research and business.

For example, researchers have shown that the "mood" of communication on twitter **reflects biological rhythms** and can even used to **predict the stock market**. A student here at UW used geocoded tweets to **plot a map of locations where "thunder" was mentioned in the context of a storm system in Summer 2012.**
Researchers from Northeastern University and Harvard University studying the characteristics and dynamics of Twitter **have an excellent resource** for learning more about this area.

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.
- analyze the relationship between location and mood based on a sample of twitter data.

This assignment is open-ended in several ways. You'll need to make some decisions about how best to solve the problem.

It is perfectly acceptable to discuss your solution on the forum, but don't share code.

Each student must submit their own solution to the problem.

You will have an unlimited number of tries for each submission.

Your code will be run in a protected environment, so you 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.

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 2.7.3 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, or can use the

class virtual machine.

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"aaaàçççñññ"
encoded_string = unicode_string.encode('utf-8')
print encoded_string
```

Getting Started

There's a video walkthrough of Problem 0 and Problem 1 available. If you are brand new to Python, or if you have trouble getting the VM set up, or if the Twitter API is causing you trouble, this video may help.
If you are new to Python, many students have recommended Google's Python class.

Problem 0: Query Twitter with Python

If you are using the class virtual machine, run the VM, open a terminal window, and use git to make sure you have the latest class materials. To edit the file in linux, you can use vi, emacs, or gedit.
Use the urllib and json libraries in python to access the basic twitter search API and return JSON data.

To retrieve recent tweets associated with the term "microsoft," you use this url:

http://search.twitter.com/search.json?q=microsoft

To access this url in Python and parse the response, you can use the following snippet:

```
import urllib
import json

response = urllib.urlopen("http://search.twitter.com/search.json?q=microsoft"
)
print json.load(response)
```

The format of the result is *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.

As you might imagine, it is fairly straightforward to convert JSON data into a Python data structure. Indeed, there is a convenient library to do so, called json, which we will use. Twitter provides only **partial documentation for understanding this data format**, but it's not difficult to deduce the structure.

Using this library, the json data is parsed and converted to a Python dictionary representing the entire result set. (If needed, take a moment to **read the documentation for Python dictionaries**). The "results" key of this dictionary corresponds holds the actual tweets; each tweet is itself another dictionary.

a) Write a program, print.py, to print out the text of each tweet in the result.

b) Generalize your program, print.py, to fetch and print 10 pages of results. Note that you can return a different page of results by passing an additional argument in the url:

```
http://search.twitter.com/search.json?q=microsoft&page=2
```

print.py should be executable in the following way:

```
$ python print.py
```

When executed, the script should print each tweet on an individual line to stdout.

What to turn in: Nothing. This is a warmup exercise.

Problem 1: Get Twitter Data

As always, the first step is to make sure your assignment materials up to date.

To access the live stream, you will need to install the **oauth2 library** so you can properly authenticate.

This library is already installed on the class virtual machine. Or you can install it yourself in your Python environment.

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

- Create a twitter account if you do not already have one.
- Go to **https://dev.twitter.com/apps** and log in with your twitter credentials.
- Click "create an application"
- Fill out the form and agree to the terms. Put in a dummy website if you don't have one you want to use.
- On the next page, scroll down and click "Create my access token"
- Copy your "Consumer key" and your "Consumer secret" into twitterstream.py
- Click "Create my access token." You can **Read more about Oauth authorization.**
- Open twitterstream.py and set the variables corresponding to the consumer key, consumer secret, access token, and access secret.

  ```
  access_token_key = "<Enter your access token key here>"
  ```

access_token_secret = "<Enter your access token secret here>"

consumer_key = "<Enter consumer key>"
consumer_secret = "<Enter consumer secret>"

- Run the following and make sure you see data flowing and that no errors occur. Stop the program with Ctrl-C once you are satisfied.

$ python twitterstream.py

You can pipe the output to a file, wait a few minutes, then terminate the program to generate a sample. Use the following command:

$ python twitterstream.py > output.txt

Let this script run for a minimum of **10 minutes**.

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 your file. You can get the first 20 lines by using the following command:

$ head -n 20 output.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 can be executed using the following command:

$ python tweet_sentiment.py <sentiment_file> <tweet_file>

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 found in a tweet, but not 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 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 "t
ab character"
  scores[term] = int(score)  # Convert the score to an integer.

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

Assume the tweet file contains data formatted the same way as the livestream data.

Your script should print to stdout the sentiment of each tweet in the file, one sentiment per line:

<sentiment:float>

NOTE: You must provide a score for **every** tweet in the sample file, even if that score is zero. However the sample file will only include English tweets

The first sentiment corresponds to the first tweet in the input file, the second sentiment corresponds to the second tweet in the input file, and so on.

Hints: The json.loads function parses a string to JSON.
Refer to the **twitter documentation** in order to determine what field to parse.

What to turn in: tweet_sentiment.py

Problem 3: Derive the sentiment of new terms

In this part you will be creating a script that computes the sentiment for the terms that **do not** appear in the file AFINN-111.txt.
Here's how you might think about the problem: We know we can use certain words to deduce the sentiment of a tweet. Once you know the sentiment of the tweet, you can assign a sentiment to the other words in the tweet. If necessary, you could repeat this process.
Don't feel obligated to use it, but the following paper may be helpful for developing a sentiment metric. Look at the Opinion Estimation subsection of the Text Analysis section in particular.

O'Connor, B., Balasubramanyan, R., Routedge, B., & Smith, N. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. (ICWSM), May 2010.

You are provided with a skeleton file, term_sentiment.py, which can be executed using the following command:

$ python term_sentiment.py <sentiment_file> <tweet_file>

Your script should print to stdout each term-sentiment pair, one pair per line, in the following format:

<term:string> <sentiment:float>

For example, if you have the pair ("foo", 103.256) it should appear in the output as:

```
foo 103.256
```

The order of your output does not matter.

What to turn in: term_sentiment.py

How we will grade Part 3: We will use a given file and make sure that your scores order the terms in roughly the same order as our solution. Your scores need not exactly match ours.
If the grader is returning "Formatting error: ", make note of the line of text returned in the message. This line corresponds to a line of your output. The grader will generate this error if line.split() does not return exactly two items. One common source of this error is to not remove the the two calls to the "lines" function in the solution template -- this function prints the number of lines in each file. Make sure to check the first two lines of your output!

Problem 4: Compute Term Frequency

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

The frequency of a term can be calculate with the following formula:
[# of occurrences of the term in all tweets]/[# of occurrences of all terms in all tweets]

frequency.py should take a file of tweets as an input and be usable in the following way:

```
$ python frequency.py <tweet_file>
```

Assume the tweet file contains data formatted the same way as the livestream data.

Your script should print to stdout each term-frequency pair, one pair per line, in the following format:

```
<term:string> <frequency:float>
```

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

```
bar 0.1245
```

Frequency measurements may take phrases into account, but this is not required. We only ask that you compute frequencies for individual tokens.

Depending on your method of parsing, you may end up with frequencies for hashtags, links, stop words, phrases, etc. Some noise is acceptable for the sake of keeping parsing simple.

What to turn in: frequency.py

Problem 5: Which State is happiest?

Write a Python script, happiest_state.py, that returns the name of the happiest state as a

string.

happiest_state.py should take a file of tweets as an input and be usable in the following way:

```
$ python happiest_state.py <sentiment_file> <tweet_file>
```

The file AFINN-111.txt contains a list of pre-computed sentiment score.

Assume the tweet file contains data formatted the same way as the livestream data.

We recommend that you build on your solution to Problem 2.

There are three different objects within the tweet that you can use to determine it's origin.

1. The coordinates object
2. The place object
3. The user object

You are free to develop your own strategy for determining the state that each tweet originates from.

Limit the tweets you analyze to those in the United States.

The live stream has a slightly different format from the response to the query you used in Problem 0. In this file, each line is a Tweet object, as **described in the twitter documentation**.

Note: Not every tweet dictionary will have a text key -- real data is dirty. Be prepared to debug, and feel free to throw out tweets that your code can't handle to get something working. For example, non-English tweets.

Your script should print the two letter state abbreviation to stdout.

Your script will not have access to the Internet, so you cannot rely on third party services to resolve geocoded locations.

What to turn in: happiest_state.py

Problem 6: Top ten hash tags

Write a Python script, top_ten.py, that computes the ten most frequently occurring hash tags from the data you gathered in Problem 1.

top_ten.py should take a file of tweets as an input and be usable in the following way:

```
$ python top_ten.py <tweet_file>
```

Assume the tweet file contains data formatted the same way as the livestream data.
In the tweet file, each line is a Tweet object, as **described in the twitter documentation**.
You should not be parsing the "text" field.

Your script should print to stdout each hashtag-count pair, one per line, in the following format:

<hashtag:string> <count:float>

For example, if you have the pair (baz, 30) it should appear in the output as:

baz 30.0

Remember your output must contain floats, not ints.

What to turn in: top_ten.py