Twitter Sentiment Analysis in Python: Instructions

Help Center

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 how Twitter can be used to analyze moods at national scale.

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

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 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"aaaÃ Ã§Ã§Ã§Ã±Ã±Ã±"
encoded_string = unicode_string.encode('utf-8')
print encoded_string
```

Getting Started

Once again: If you are new to Python, many students have recommended Google's Python class.

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, but you can install it yourself in your Python environment. (The command `$ pip install oauth2` should work for most environments.)

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.
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.

9. If you wish, modify the file to use the twitter search API to search for specific terms. For example, to search for the term "microsoft", you can pass the following url to the twitterreq function:

```
https://api.twitter.com/1.1/search/tweets.json?q=microsoft
```

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 `problem_1_submission.txt` by using the following command:

```
$ head -n 20 output.txt > problem_1_submission.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. You can run the skeleton program like this:

```
$ python tweet_sentiment.py AFINN-111.txt output.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.

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 `output.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. Once again: **The nth line of 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.

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.

Hint: 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!

What to turn in: The file `tweet_sentiment.py` after you've verified that it returns the correct answers.

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 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 `soccer` always appears in proximity with positive words like `great` and `fun`, then we can deduce that the term `soccer` itself carries a positive sentiment.

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 output to stdout. Each line of output should contain a term, followed by a space, followed by the sentiment. That is, each line should be in the format `<term:string> <sentiment:float>`

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

```
foo 103.256
```

The order of your output does not matter.

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

How we will grade Part 3: We will run your script on a file that contains strongly positive and strongly negative tweets and verify that the non-sentiment-carrying terms in the strongly positive tweets are assigned a higher score than the non-sentiment-carrying terms in negative tweets.Â Your scores need not (and likely will not) exactly match any specific solution.

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 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 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 frequency.py <tweet_file>
```

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

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 (and the frequency will be at least 25!) 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.

What to turn in: The file `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.

Your script `happiest_state.py` should take a file of tweets as input. It will be called from the command line like this:

```
$ 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.

It's a good idea to make use of your solution to Problem 2.

There are different ways you might assign a location to a tweet. Here are three:

- Use the `coordinates` field (a part of the `place` object, if it exists, to geocode the tweet. This method gives the most reliable location information, but unfortunately this field is not always available and you must figure out some way of translating the coordinates into a state.
- Use the other metadata in the `place` field. Much of this information is hand-entered by the twitter user and may not always be present or reliable, and may not typically contain a state name.
- Use the `user` field to determine the twitter user's home city and state. This location does not necessarily correspond to the location where the tweet was posted, but it's reasonable to use it as a proxy.

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

You may find it useful to use this python dictionary of state abbreviations.

You can ignore any tweets for which you cannot assign a location in the United States.

In this file, each line is a Tweet object, asdescribed in the twitter documentation.

Note: Not every tweet will have a `text` field --- again, 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.Â For example, you might choose to ignore all non-English tweets.

Your script should print the two letter state abbreviation of the state with the highest average tweet sentiment to stdout.

Note that you may need a **lot** of tweets in order to get enough tweets with location data. Let the live stream run for a while if you wish.

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: The file `happiest_state.py`

Problem 6: Top ten hash tags

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

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

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

You should 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. To find the hashtags, you should not parse the `text` field; the hashtags have already been extracted by twitter.

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

Your script should print output to stdout. Each line of output should contain a hashtag, followed by a space, followed by the frequency of that hashtag in the **entire file**. There should be one line per **unique** hashtag in the entire file. Each line should be in the format `<hashtag:string> <frequency:float>`

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

```
bar 30
```

What to turn in: the file `top_ten.py`