Twitter Data Wrangling: Project 3

{aaronweiss, ilevyor, jholzman, jreedie, kvedder, poddarh, timothymcnam}@umass.edu
gordon@cs.umass.edu
February 13, 2017

1. Description

In this assignment, you will be analyzing tweets from Donald J. Trump using Python list functions. This task will be your first real exercise in functional programming with Python. It will require you to make extensive use of built-in functions like `map`, `filter`, and `reduce`.

2. Objectives

1. Learn how to program functionally in Python.
2. Make extensive use of Python built-in functions.
3. Understand the basics of data analysis.

3. Getting Started

To get started, you should create a new project with a single file named `analyze.py`. This file will be where you implement all of the data analytics functions described in the rest of the document. You'll also need to place the included `tweets.json` into the root folder of your project. This is where you will load the tweets for analysis. Don't worry if you don't know what a `json` file is as our code handles those details. You can use the following template for `analyze.py` to get started. It includes some simple code to load the `tweets.json` globally which you may find useful during testing, and what we will use for grading.

```
#!/usr/bin/env python3
from functools import *
import json

with open("tweets.json", "r") as tweet_db:
    tweets = json.load(tweet_db)

# TODO: implement assigned functions
```

Before you start, you'll also want to know a bit about the input format. Any function described as taking `xs` or `ys` will take a list of arbitrary elements. Functions described as taking a `tweet` or `tweets` will take either a single tweet or a list of tweets where tweets are Python dictionaries defined like so:

```
tweet = {
  "username": "@realDonaldTrump",
  "source": "Twitter for iPhone",
  "content": "This is totally a real tweet.",
  "favorites": 11,
  "retweets": 5,
}
```

4. Baby Steps

In this section of the assignment, you will get started by writing a number of basic functions. It is important to get these right because they will be used heavily throughout the assignment and in both testing and grading.

`def flatten(xs)` – Flattens a list of lists to simply a list.

Example Uses:

```
flatten([[1, 2], [], [3, 4]]) # => [1, 2, 3, 4]
```

`def difference(xs, ys)` – Finds all the elements that are in either `xs` or `ys`, but not both.

Example Uses:

```
difference([1, 2], [2, 3]) # => [1, 3]
```

`def to_text(tweets)` – Converts from a list of tweets to a list of tweet contents.

Example Uses:

```
tweet1 = {"content": "My hands are YUUGE!", ...}
tweet2 = {"content": "#MAGA", ...}
tweet3 = {"content": "I hate Little Marco.", ...}
tweets = [tweet1, tweet2, tweet3]
to_text(tweets) # => ["My hands are YUUGE!", "#MAGA", "I hate Little Marco."]
```

`def to_lowercase(tweets)` – Converts the content of each tweet in the list of tweets to lowercase.

Example Uses:

```
tweet1 = {"content": "My hands are YUUGE!", ...}
tweet2 = {"content": "#MAGA", ...}
tweet3 = {"content": "I hate Little Marco.", ...}
tweets = [tweet1, tweet2, tweet3]
to_text(to_lowercase(tweets))
# => ["my hands are yuuge!", "#maga", "i hate little marco."]
```

`def nonempty(tweets)` – Removes all tweets with empty contents from the list of tweets.

Example Uses:

```
tweet1 = {"content": "#MAGA", ...}
tweet2 = {"content": "", ...}
tweet3 = {"content": "#CrookedHillary", ...}
tweets = [tweet1, tweet2, tweet3]
to_text(nonempty(tweets)) # => ["#MAGA", "#CrookedHillary"]
```

`def total_word_count(tweets)` – Calculates the total number of words in each tweet in a list.

Example Uses:

```
tweet1 = {"content": "I am President now.", ...}
tweet2 = {"content": "Make America Safe Again!", ...}
tweet3 = {"content": "Make America Great Again!", ...}
tweets = [tweet1, tweet2, tweet3]
total_word_count(tweets) # => 12
```

`def hashtags(tweet)` – Gets a list of all the hashtags from the specified tweet.

Example Uses:

```
tweet = {"content": "Hello, America. #MAGA #Trump2016", ...}
hashtags(tweet) # => ["#MAGA", "#Trump2016"]
```

`def mentions(tweet)` – Gets a list of all the mentions from the specified tweet.

Example Uses:

```
tweet = {"content": "@aatxe would be a better president than me.", ...}
mentions(tweet) # => ["@aatxe"]
```

5. Counting Problems

In this section of the assignment, you will use built-in functions in combination with some of the previously defined functions to answer a number of collection and counting problems. Some of these functions might be trickier than the earlier ones, so tread carefully!

`def all_hashtags(tweets)` – Returns a list of all hashtags from a list of tweets.

Example Uses:

```
tweet1 = {"content": "Hello, America. #MAGA #Trump2016", ...}
tweet2 = {"content": "Lock her up! #CrookedHillary #MAGA", ...}
tweet3 = {"content": "No hashtags in here.", ...}
tweets = [tweet1, tweet2, tweet3]
all_hashtags(tweets) # => ["#MAGA", "#Trump2016", "#CrookedHillary", "#MAGA"]
```

`def all_mentions(tweets)` – Returns a list of all mentions from a list of tweets.

Example Uses:

```
tweet1 = {"content": "@cnn is fake news!", ...}
tweet2 = {"content": "Can't believe how many people buy @cnn fake news!", ...}
tweet3 = {"content": "@marcorubio and I are friends now!", ...}
tweets = [tweet1, tweet2, tweet3]
all_mentions(tweets) # => ["@cnn", "@cnn", "@marcorubio"]
```

`def all_caps_tweets(tweets)` – Returns a list of all tweets that are completely capitalized from the given list of tweets.

Example Uses:

```
tweet1 = {"content": "MAKE AMERICA SAFE AGAIN!", ...}
tweet2 = {"content": "Can't believe how many people buy @cnn fake news!", ...}
tweet3 = {"content": "AMERICA FIRST!", ...}
tweets = [tweet1, tweet2, tweet3]
to_text(all_caps_tweets(tweets))
# => ["MAKE AMERICA SAFE AGAIN!", "AMERICA FIRST!"]
```

`def count_individual_words(tweets)` – Counts all the words in all the tweets and returns the count as a dictionary mapping each word to its count.

Example Uses:

```
tweet1 = {"content": "MAKE AMERICA SAFE AGAIN!", ...}
tweet2 = {"content": "america", ...}
tweet3 = {"content": "AMERICA FIRST!", ...}
tweets = [tweet1, tweet2, tweet3]
count_individual_words(tweets)
# => {
#   "MAKE": 1,
#   "AMERICA": 2,
#   "SAFE": 1,
#   "AGAIN!": 1,
#   "FIRST!": 1,
#   "america": 1,
# }
```

`def count_individual_hashtags(tweets)` – Counts all the hashtags in all the tweets and returns the count as a dictionary mapping each word to its count.

Example Uses:

```
tweet1 = {"content": "#MAGA #Trump2016", ...}
tweet2 = {"content": "#MakeAmericaGreatAgain", ...}
tweet3 = {"content": "No hashtags in here.", ...}
tweets = [tweet1, tweet2, tweet3]
count_individual_hashtags(tweets)
# => {
#   "#MAGA": 1,
#   "#Trump2016": 1,
#   "#MakeAmericaGreatAgain": 1,
# }
```

`def count_individual_mentions(tweets)` – Counts all the mentions in all the tweets and returns the count as a dictionary mapping each word to its count.

Example Uses:

```
tweet1 = {"content": "@cnn is fake news!", ...}
tweet2 = {"content": "Can't believe how many people buy @cnn fake news!", ...}
tweet3 = {"content": "@marcorubio and I are friends now!", ...}
tweets = [tweet1, tweet2, tweet3]
count_individual_mentions(tweets)
# => {
#   "@cnn": 2,
#   "@marcorubio": 1,
# }
```

`def n_most_common(n, word_count)` – Calculates the `n` most common keys in `word_count`, sorted from most to least common and sorted in alphabetical order when the number of occurrences is the same.

Example Uses:

```
tweet1 = {"content": "MAKE AMERICA SAFE AGAIN!", ...}
tweet2 = {"content": "america", ...}
tweet3 = {"content": "AMERICA FIRST!", ...}
tweets = [tweet1, tweet2, tweet3]
n_most_common(1, count_individual_words(tweets)) # => [("AMERICA", 2)]
```

6. Simple Filters

In this section of the assignment, you will write some basic filters that can be used to separate the data by source, and identify patterns after partitioning. This section was inspired by an article in Variance Explained analyzing Trump's tweets. The piece found that tweets from Android are written by Trump himself, and tweets from iOS are written by his staffers. With these filters, you can replicate a bit of that analysis yourself!

`def iphone_tweets(tweets)` – Filters a list of tweets to find only those made from an iPhone.

Example Uses:

```
tweet1 = {"content": "My hands are YUUGE!", "source": "Twitter for iPhone", ...}
tweet2 = {"content": "#MakeAmericaOkAgain", "source": "Twitter for Android", ...}
tweet3 = {"content": "#CrookedHillary", "source": "Twitter for iPhone", ...}
tweets = [tweet1, tweet2, tweet3]
to_text(iphone_tweets(tweets)) # => ["My hands are YUUGE!", "#CrookedHillary"]
```

`def android_tweets(tweets)` – Filters a list of tweets to find only those made from an Android device.

Example Uses:

```
tweet1 = {"content": "My hands are YUUGE!", "source": "Twitter for iPhone", ...}
tweet2 = {"content": "#MakeAmericaOkAgain", "source": "Twitter for Android", ...}
tweet3 = {"content": "I hate Little Marco.", "source": "Twitter for iPhone", ...}
tweets = [tweet1, tweet2, tweet3]
to_text(android_tweets(tweets)) # => ["#MakeAmericaOkAgain"]
```

7. Statistical Analysis

In this section of the assignment, you will implement a number of statistical operations that will help you to identify patterns in tweets based off their popularity. These operations require some knowledge of basic statistics to understand, but you should be able to implement them based on these instructions.

`def average_favorites(tweets)` – Computes the average number of favorites from the list of tweets, rounding appropriately.

Example Uses:

```
tweet1 = {"favorites": 32, ...}
tweet2 = {"favorites": 8, ...}
tweet3 = {"favorites": 16, ...}
tweets = [tweet1, tweet2, tweet3]
average_favorites(tweets) # => 19
```

`def average_retweets(tweets)` – Computes the average number of retweets from the list of tweets, rounding appropriately.

Example Uses:

```
tweet1 = {"retweets": 32, ...}
tweet2 = {"retweets": 80, ...}
tweet3 = {"retweets": 16, ...}
tweets = [tweet1, tweet2, tweet3]
average_retweets(tweets) # => 43
```

`def sort_by_favorites(tweets)` – Sorts the tweets by the number of favorites they have.

Example Uses:

```
tweet1 = {"favorites": 32, ...}
tweet2 = {"favorites": 8, ...}
tweet3 = {"favorites": 16, ...}
tweets = [tweet1, tweet2, tweet3]
sort_by_favorites(tweets) # => [tweet2, tweet3, tweet1]
```

`def sort_by_retweets(tweets)` – Sorts the tweets by the number of retweets they have.

Example Uses:

```
tweet1 = {"retweets": 32, ...}
tweet2 = {"retweets": 8, ...}
tweet3 = {"retweets": 16, ...}
tweets = [tweet1, tweet2, tweet3]
sort_by_retweets(tweets) # => [tweet2, tweet3, tweet1]
```

`def upper_quartile(tweets)` – Assuming the input is sorted, find the tweet representative of the upper quartile. This is the tweet representing the 75th percentile in the characteristic the list has been sorted by. You can compute it as 3/4th of the size of the input.

Example Uses:

```
tweet1 = {"favorites": 32, ...}
tweet2 = {"favorites": 8, ...}
tweet3 = {"favorites": 16, ...}
tweet4 = {"favorites": 19, ...}
tweet5 = {"favorites": 44, ...}
tweets = [tweet1, tweet2, tweet3, tweet4, tweet5]
upper_quartile(sort_by_favorites(tweets)) # => tweet4
```

`def lower_quartile(tweets)` – Assuming the input is sorted, find the tweet representative of the lower quartile. This is the tweet representing the 25th percentile in the characteristic the list has been sorted by. You can compute it as 1/4th of the size of the input.

Example Uses:

```
tweet1 = {"retweets": 32, ...}
tweet2 = {"retweets": 8, ...}
tweet3 = {"retweets": 16, ...}
tweet4 = {"retweets": 19, ...}
tweet5 = {"retweets": 44, ...}
tweets = [tweet1, tweet2, tweet3, tweet4, tweet5]
lower_quartile(sort_by_retweets(tweets)) # => tweet2
```

`def top_quarter_by(tweets, factor)` – Assuming the input is sorted by `factor`, find all tweets with `factor` greater than or equal to the upper quartile representative found using the `upper_quartile` function you've implemented.

Example Uses:

```
tweet1 = {"retweets": 32, ...}
tweet2 = {"retweets": 8, ...}
tweet3 = {"retweets": 16, ...}
tweet4 = {"retweets": 19, ...}
tweet5 = {"retweets": 44, ...}
tweets = [tweet1, tweet2, tweet3, tweet4, tweet5]
top_quarter_by(sort_by_retweets(tweets), "retweets") # => [tweet4, tweet5]
```

`def bottom_quarter_by(tweets, factor)` – Assuming the input is sorted by `factor`, find all tweets with `factor` less than or equal to the lower quartile representative found using the `lower_quartile` function you've implemented.

Example Uses:

```
tweet1 = {"favorites": 32, ...}
tweet2 = {"favorites": 8, ...}
tweet3 = {"favorites": 16, ...}
tweet4 = {"favorites": 19, ...}
tweet5 = {"favorites": 44, ...}
tweets = [tweet1, tweet2, tweet3, tweet4, tweet5]
bottom_quarter_by(sort_by_favorites(tweets), "favorites") # => [tweet1, tweet2]
```

8. Submitting

If you've come this far, you've completed the project. Congratulations! We hope you've taken the opportunity already to experiment with the code you've written, and tested that it works! All that's left then is for you to prepare the submission archive and upload it to Moodle. To help with this, we included a `submit.py` script that you can run to build the archive. To use it, either run it directly from the project directory, or pass it a path to the project like so `./submit.py path/to/hw3`. It should produce a timestamped submission archive in the folder you're running it from. If you need any help with this, just make a post on Piazza! Once again, congratulations on completing the assignment!