Working with Tweets

This is part 2 of the Python portion of a Software Carpentry workshop that is built around some light handling of Twitter data. There are four parts. At the end of this section participants will have produced a chart similar to that from the first section but which is the result of processing Twitter data. Skills covered include:

- Planning a program
- Reading data from files
- Tokenization with the nltk
- Tweet preprocessor

With the basics out of the way we can move to using them to work with a specific data type: Tweets. While perhaps not a truely representative sample of what is important Twitter does offer an important snapshot into a variety of communities and trends. In this portion of the workshop we'll focus on handling some pre-collected tweets while drawing on our histogram skills to see what words are most strongly associated with the chosen topic.

The topic? We'll go with something likely to be of general interest: Trump and Clinton in the US Presidential Primaries.

Planning the Program

Let's start by opening "Python-Lesson2.ipynb".

In the top cell we'll start by planning out our program with comments:

```
# This program will read twitter data stored in a file line by line and then produce
a graph showing the most common words used in the tweets.

# Load twitter data from a file

# Process that data to extract the tweet text and build a list of words

# Produce the chart
```

As always there will be sub steps but this is enough to get us started for now.

Reading data from a file

Let us play with opening files in a new cell:

```
open("trumpTweets.json", "r")
```

When we run this we'll see output like the following:

```
<_io.TextIOWrapper name='trumpTweets.json' mode='r' encoding='UTF-8'>
```

What we have done is create a connection to a file and what is displayed are the details of this connection. What we have failed to do is capture this connection with a variable so that we can work with the file. This is easily remedied by simply assigning the output of the open file function to a variable

```
inputFile = open("trumpTweets.json", "r")
```

Before we go any further we need to make sure that we close the file too. Python is pretty good about doing this for us but hiccups happen and when they do they can result in data corruption, especially when we are writing ("w") or appending ("a"). So let's close the file at the end of the cell block:

```
inputFile = open("trumpTweets.json", "r")

inputFile.close()
```

We'll just remember this line is here and not display it further.

If we run this line we get nothing returned. If we print inputFile we get the same file info as before. What we need to do is unpack the file line by line (we don't always have to unpack the file line by line but it is good practice for the time being). There is a lot of data in 90 tweets so let's just read one line for now using the readline() method for python file objects:

```
inputFile = open("trumpTweets.json", "r")
print(inputFile.readline())

{"is_quote_status": true, "in_reply_to_status_id_str": null, "coordinates": null, "po
ssibly_sensitive": false, "in_reply_to_screen_name": null, "lang": "en", "favorite_co
unt": 0, "favorited": false, "metadata": {"iso_language_code": "en", "result_type": "
recent"}, "entities": {"hashtags": [{"text": "Trump", "indices": [20, 26]}], "symbols
": [], "urls": [{"expanded_url": "https://twitter.com/realdonaldtrump/status/72552425
7883697152", "display_url": "twitter.com/realdonaldtrum\u2026", "indices": [139, 140]
, "url": "https://t.co/Nayyhhonr9"}], "user_mentions": [{"id": 3314758074, "indices":
 [3, 18], "name": "JP Moore (REAL)", "id_str": "3314758074", "screen_name": "Campaign
_Trump"}]}, "source": "<a href=\"http://twitter.com/download/android\" rel=\"nofollow
\">Twitter for Android</a>", "created_at": "Thu Apr 28 04:23:46 +0000 2016", "retweet
ed": false, "in_reply_to_user_id_str": null, "text": "RT @Campaign_Trump: #Trump bill
ionaire is staying at Holiday Express. It's shows you he is just a nice plain no BS d
own to earth guy.  http\u2026", "retweet_count": 11, ETC.
```

Things are looking good, but what a blob of text! What we are seeing is JSON formatted data. If it looks a lot like a dictionary that's because that's pretty much what JSON is as far as we are concerned. That means that even though it looks nasty if you know the format and the keys you can quickly grab any content you want. The body of a tweet is associated with the key "text" so let's grab that for the first tweet:

```
inputFile=open("trumpTweets.json", "r")
line = inputFile.readline()
line["text"]
```

But this throws an error:

```
TypeError: string indices must be integers
```

The problem is that while the line has the potential to be a dictionary python is currently reading it just as a string. We need to import the json library and tell python to load the line as json:

```
import json
inputFile=open("trumpTweets.json", "r")
line = json.loads(inputFile.readline())
line["text"]
```

Now we get the text body of the first tweet.

Before we go further let's use the fact that we now have actual json/dictionary data to clean up the presentation of the entire line:

```
print(json.dumps(line, indent = 4))
```

While `loads` ingests data in the json format `dumps` pushes it out. Note that the 's' on the end isn't a pluralization, it signifies that we are working with *strings*. If we wanted to work with raw file objects then we would remove it (and need to change lots of things about how we interact with the data so we'll leave the s on).

We could go on with this method of loading data but it isn't really safe. If our program crashes between the open and the close there is no automatic clean-up and close tool to try and prevent data corruption. A better method is `with`:

```
with open("trumpTweets.json", "r") as inputFile:
    line = json.loads(inputFile.readline())
    line["text"]
```

Similar setup, safer practice.

With this new setup let us print the text of all the tweets

```
with open("trumpTweets.json", "r") as inputFile:
    for line in inputFile:
        tweet = json.loads(line)
        print(tweet["text"])
```

> Finish off loading the data by copying what we have up to our top cell and modifying it to load each line into a list called tweets.

```
tweets = []
with open("trumpTweets.json", "r") as inputFile:
    for line in inputFile:
        tweets.append(json.loads(line))
```

Tokenize the tweets

Now that we can load our data the next step is to split the body text of the tweets in words so that we can count them. Just cutting at spaces (`.split()`) won't do what we need since there is some punctuation we need to keep and some we need to chop off. What we need is a tokenization tool. It shouldn't surprise anyone that the standard tools aren't sure how to handle twitter conventions but we'll be able to work around this by borrowing a hack.

Let's see what happens when we bring in the tokenization tool from the NLTK (Natural Language Toolkit, a powerful NLP library for Python that comes with Anaconda).

> The NLTK is included in the Anaconda installer. If people are missing it though then this is an opportunity to show them how to search for and install software for Anaconda from the command line. Use `conda search nltk` and then `conda install -c anaconda nltk=3.2.2`.
>
> It should be noted that there are a great many corpuses that can be accessed via the nltk. This can be done by issuing `nltk.download()`. Unlike many commands this one will bring up an interactive window. If you are working with the nltk and feel you are missing something that examples online are taking for granted this is likely the way to get it.

```
from nltk.tokenize import word_tokenize

fake_tweet = "RT @symulation: just an example I'd like to share! :D http:// example.c
om #NLP #NLTK"
print(word_tokenize(fake_tweet))
```

Which churns out the terrible response:

```
['RT', '@', 'symulation', ':', 'just', 'an', 'example', 'I', "'d", 'like', 'to', 'sha
re', '!', ':', 'D', 'http', ':', '//example.com', '#', 'NLP', '#', 'NLTK']
```

The problem is that Tweets have some pretty non-standard syntax so this isn't as easy as simply splitting a string at the spaces so we'll need to pull a tool to help us out. That tool is a pre-written tweet preprocessor. We'll look it over quickly but mostly accept it and set aside the details for now. Run the provided text pre-processor in your notebook:

```
#Instantiate a text pre-processor
import re

emoticons_str = r"""
    (?:
        [:=;] # Eyes
        [oO\-]? # Nose (optional)
        [D\)\]\(\]/\\OpP] # Mouth
    )"""

regex_str = [
    emoticons_str,
    r'<[^>]+>', # HTML tags
    r'(?:@[\w_]+)', # @-mentions
    r"(?:\#+[\w_]+[\w\'_\-]*[\w_]+)", # hash-tags
    r'http[s]?://(?:[a-z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-f][0-9a-f]))+', # URLs

    r'(?:(?:\d+,?)+(?:\.?\d+)?)', # numbers
    r"(?:[a-z][a-z'\-_]+[a-z])", # words with - and '
    r'(?:[\w_]+)', # other words
    r'(?:\S)' # anything else
]

tokens_re = re.compile(r'('+'|'.join(regex_str)+')', re.VERBOSE | re.IGNORECASE)
emoticon_re = re.compile(r'^'+emoticons_str+'$', re.VERBOSE | re.IGNORECASE)

def tokenize(s):
    return tokens_re.findall(s)

def preprocess(s, lowercase=False):
    tokens = tokenize(s)
    if lowercase:
        tokens = [token if emoticon_re.search(token) else token.lower() for token in
tokens]
    return tokens
```

Once we have run the block with the pre-processor we now have access to the preprocess function in all the cells of the notebook. We modify the previous tokenization as follows:

```
from nltk.tokenize import word_tokenize

fake_tweet = "RT @symulation: just an example I'd like to share! :D http:// example.c
om #NLP #NLTK"
print(preprocess(fake_tweet))
```

Nice. Now hashtags and at-mentions stay together. Same with emoticons.

> Move this tokenization into our program at the top, modifying it so that rather than saving each tweet to the list of tweets it prints out the tokenization of each tweet.

```
with open("trumpTweets.json", "r") as inputFile:
    for line in inputFile:
        tweet=(json.loads(line))
        print(preprocess(['text']))
```

Or as a single line:

```
with open("trumpTweets.json", "r") as inputFile:
    for line in inputFile:
        print(preprocess(json.loads(line)['text']))
```

Adding a counter

We could use our histogram at this point but there is a library for counters that gives us some rather nice features, like automatically counting members of a list.

```
from collections import Counter

with open("trumpTweets.json", "r") as inputFile:
    count_all = Counter()
    for line in inputFile:
        tweet=json.loads(line)
        terms = [term for term in preprocess(tweet['text'])]
        count_all.update(terms)
    # Print the first 5 most frequent words
    print(count_all.most_common(5))
```

Note the fancy move where we assign content to terms. Here we are taking what would otherwise be a multi-line for loop and compressing it into a single line. The "term" immediately after the opening square bracket is not the value that is actually passed to terms. Rather, it is the list of all such terms that is passed. Very slick.

Despite our fancy move the tool still prints out a lot of junk though.

```
[('.', 73), ('#Trump', 70), (':', 70), ('RT', 63), ('…', 45)]
```

We need a stopword list. The NLTK has one that is easy to import. We also need to strip punctuation so we'll

need the string library, modifying the file as follows:

```
from collections import Counter
from nltk.corpus import stopwords
import string

punctuation = list(string.punctuation)
stop = stopwords.words('english') + punctuation + ['rt', 'via', 'RT', '…']


with open("trumpTweets.json", "r") as inputFile:
    count_all = Counter()
    for line in inputFile:
        tweet=json.loads(line)
        terms = [term for term in preprocess(tweet['text']) if term not in stop]
        count_all.update(terms)
    # Print the first 5 most frequent words
    print(count_all.most_common(5))
```

This is pretty elegant. There is still some junk output though.

> Change what we have written so far to remove the terms you don't want, like "amp" (short for "ampersand").

Note that this kind of tuning is inevitable and happens on a task by task basis.

Plotting

Once we have the data as we want it is time to look at plotting a chart of top words (Note that with a few tweaks we could be charting users with the most tweets, tweets with the most likes, etc.). We'll use a snippet in the notebook to give us a boost.

```
# Print the current version of the chart
%matplotlib inline

count_all_dict = dict(count_all.most_common(5))
import matplotlib.pyplot as plt
plt.figure(figsize=(15, 6))
plt.bar(range(len(count_all_dict)), count_all_dict.values(), align='center')
plt.xticks(range(len(count_all_dict)), list(count_all_dict.keys()),rotation='vertical
')

plt.show()
```

> what happens when we modify or even remove the plt.figure line?

Try it.

> how does range(len(count*all*dict)) work?