Advanced Data Science I Project

Ben Barrett

September 29, 2017

R Markdown

I will be completing the project focused on pulling social media data to identify times and areas hardest hit by Hurricane Harvey.

Scraping Data

I will be using Twitter as my social media platform, and will rely on a combination of the Twitter API and online tutorials to help me with scraping the data. The first step is to gain access to the Twitter API, which I have done by following this tutorial. Here is the link to my Twitter app page.

Next, I needed to download and install the Python library Tweepy, in order to connect to Twitter Streaming API and download data. I have done this by following these installation instructions. Note: Python and pip must be installed first (For pip: Must type 'python -m pip XXXX' (without quotes) in order for the command to work). Next, I created a Python program in Notepad++ called twitter_streaming.py, based off of these instructions, and have started my stream searching for the hashtags 'HurricaneHarvey' and 'HurricaneHarveyRelief'. I am exporting the output to a JSON file, twitter_data.json, which I will later have to parse for relevant data.

```
#twitter_streaming.py

#Import the necessary methods from tweepy library
import tweepy
from tweepy import Stream
from tweepy.streaming import StreamListener
from tweepy import OAuthHandler
import json

#Variables that contains the user credentials to access Twitter API
access_token = "Access_Token"
access_token_secret = "Access_Token_Secret"
consumer_key = "Consumer_Key"
consumer_secret = "Consumer_Secret"

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth)
@classmethod
def parse(cls, api, raw):
    status = cls.first_parse(api, raw)
    setattr(status, 'json', json.dumps(raw))
    return status

# Status() is the data model for a tweet
tweepy.models.Status.first_parse = tweepy.models.Status.parse
tweepy.models.Status.parse = parse
class MyListener(StreamListener):
```

```
    def on_data(self, data):
        try:
            with open('twitter_data.json', 'a') as f:
                f.write(data)
                return True
        except BaseException as e:
            print("Error on_data: %s" % str(e))
        return True

    def on_error(self, status):
        print(status)
        return True

#Set the hashtag to be searched
twitter_stream = Stream(auth, MyListener())
twitter_stream.filter(track=['HurricaneHarvey', 'HurricaneHarveyRelief'])
```

I began the Twitter data stream at 9:10AM on 9/1/2017. I stopped the Twitter data stream at 9:56AM on 9/4/2017. Program run time = 36 hours, 46 minutes. A total of 3,289,336 KB (3.290 GB) of Twitter data was collected, which corresponds to 1,491.086 KB of data per minute, on average.

Analyzing Data

First, I streamed in the Twitter JSON file using a handler to randomly sample 10% of each page of JSON lines.

```
library(jsonlite)
JSONpath <- "C:\\Program Files (x86)\\Python\\Python36\\twitter_data.json"
con_out <- file(tmp <- tempfile(), open = "wb")
stream_in(file(JSONpath), handler= function(randsam){
  randsam <- randsam[sample(1:nrow(randsam), round(0.1*length(randsam))),]
  stream_out(randsam, con_out, pagesize=1000)
}, pagesize=500, verbose=TRUE)
close(con_out)

tweets <- stream_in(file(tmp))
nrow(tweets)
unlink(tmp)
```

420,267 records were found, which corresponds to the 420,267 tweets found during the initial stream. Then, after the random sample, 10,079 tweets were read into R.

Now I need to do some processing of the JSON file, 'tweets', just read into R. I also converted the flattened dataframe, 'tweets_flat', back to a JSON object - and later a JSON file - to be run in the Python script below.

```
tweets_flat <- flatten(tweets)
coordinates <- tweets_flat$place.bounding_box.coordinates
tweets_json <- toJSON(tweets_flat)
write(tweets_json, "C:\\Program Files (x86)\\Python\\Python36\\tweets_json.json")
```

I then created a figure to help visualize on what days the tweets were collected from, and how many tweets came from each day.

I next created another Python program called tweet_geolocator.py, again based off of these instructions.

```
# tweet_geolocator.py
import json
# Tweets are stored in the file "fname". In the file used for this script,
# each tweet was stored on one line
fname = 'tweets_json.json'
with open(fname, 'r') as f:

    #Create dictionary to later be stored as JSON. All data will be included
    # in the list 'data'
    users_with_geodata = {
        "data": []
    }
    all_users = []
    total_tweets = 0
    geo_tweets  = 0
    for line in f:
        tweet = json.loads(line)
        if tweet['user']['id']:
            total_tweets += 1
            user_id = tweet['user']['id']
            if user_id not in all_users:
                all_users.append(user_id)

                #Give users some data to find them by. User_id listed separately
                # to make iterating this data later easier
                user_data = {
                    "user_id" : tweet['user']['id'],
                    "features" : {
                        "name" : tweet['user']['name'],
                        "id": tweet['user']['id'],
                        "screen_name": tweet['user']['screen_name'],
                        "tweets" : 1,
                        "location": tweet['user']['location'],
                    }
                }
                #Iterate through different types of geodata to get the variable primary_geo
                if tweet['coordinates']:
                    user_data["features"]["primary_geo"] = str(tweet['coordinates']['coordinates']
                    user_data["features"]["geo_type"] = "Tweet coordinates"
                elif tweet['place']:
                    user_data["features"]["primary_geo"] = tweet['place']['full_name'] + ", " + tweet['
                    user_data["features"]["geo_type"] = "Tweet place"
                else:
                    user_data["features"]["primary_geo"] = tweet['user']['location']
                    user_data["features"]["geo_type"] = "User location"
                #Add only tweets with some geo data to .json. Comment this if you want to include all t
                if user_data["features"]["primary_geo"]:
                    users_with_geodata['data'].append(user_data)
                    geo_tweets += 1

            #If user already listed, increase their tweet count
            elif user_id in all_users:
                for user in users_with_geodata["data"]:
```

```
                    if user_id == user["user_id"]:
                        user["features"]["tweets"] += 1

    #Count the total amount of tweets for those users that had geodata
    for user in users_with_geodata["data"]:
        geo_tweets = geo_tweets + user["features"]["tweets"]
    #Get some aggregated numbers on the data
    print("The file included ", str(len(all_users)), " unique users who tweeted with or without geo data
    print("The file included ", str(len(users_with_geodata['data'])), " unique users who tweeted with g
    print("The users with geo data tweeted ", str(geo_tweets), " out of the total", str(total_tweets),
# Save data to JSON file
with open('Harvey_tweets_geo.json', 'w') as fout:
    fout.write(json.dumps(users_with_geodata, indent=4))
```

The next step is to geocode my Twitter data and exclude all tweets that were made from outside the area impacted by Hurricane Harvey up through 9:56AM on 9/4/2017. Geocoding the tweets will primarily rely on the user location (entered once when a user first starts Twitter), rather than the location associated with an individual tweet, as most users turn the tweet location data off. In this case, however, this should work to my advantage - as people who fled the impacted area before the storm hit should still have their user location linked to the impacted area. This mode of analysis relies on the assumption that the more tweets about Hurricane Harvey coming out of an area, the worse off that area is.

After the tweets are geocoded and restricted, I will visualize using ggplot2 in R. I will also run spatial statistics to quantify the spatial intensity and clustering of tweets. Through this method I will be able to visualize areas with a lot of Hurricane Harvey tweets coming out of them, and quantify - with statistical significance - if tweets are clustering in an area more than would be expected under complete spatial randomness. I can also compare the clustering of different areas, and quantify how much more one area may have tweet clustering as compared to another. Finally, this data can be stratified by days and times, and several maps can be created to 'track' the progress of the storm. As a final check, the Twitter heatmaps can be compared to meteorological maps for the same overlapping dates and times, and the accuracy of tracking the storm via Twitter can be assessed.

```
## Session info -------------------------------------------------------------

##  setting  value
##  version  R version 3.3.2 (2016-10-31)
##  system   x86_64, mingw32
##  ui       RTerm
##  language (EN)
##  collate  English_United States.1252
##  tz       America/New_York
##  date     2017-09-29

## Packages -----------------------------------------------------------------

##  package   * version date       source
##  backports   1.0.5   2017-01-18 CRAN (R 3.3.2)
##  base      * 3.3.2   2016-10-31 local
##  datasets  * 3.3.2   2016-10-31 local
##  devtools    1.13.3  2017-08-02 CRAN (R 3.3.3)
##  digest      0.6.12  2017-01-27 CRAN (R 3.3.3)
##  evaluate    0.10    2016-10-11 CRAN (R 3.3.3)
##  graphics  * 3.3.2   2016-10-31 local
##  grDevices * 3.3.2   2016-10-31 local
##  htmltools   0.3.5   2016-03-21 CRAN (R 3.3.3)
##  knitr       1.17    2017-08-10 CRAN (R 3.3.3)
```

```
##  magrittr    1.5     2014-11-22 CRAN (R 3.3.3)
##  memoise     1.1.0   2017-04-21 CRAN (R 3.3.3)
##  methods   * 3.3.2   2016-10-31 local
##  Rcpp        0.12.9  2017-01-14 CRAN (R 3.3.2)
##  rmarkdown   1.6     2017-06-15 CRAN (R 3.3.3)
##  rprojroot   1.2     2017-01-16 CRAN (R 3.3.3)
##  stats     * 3.3.2   2016-10-31 local
##  stringi     1.1.3   2017-03-21 CRAN (R 3.3.3)
##  stringr     1.2.0   2017-02-18 CRAN (R 3.3.3)
##  tools       3.3.2   2016-10-31 local
##  utils     * 3.3.2   2016-10-31 local
##  withr       2.0.0   2017-07-28 CRAN (R 3.3.3)
##  yaml        2.1.14  2016-11-12 CRAN (R 3.3.3)
```