MARCH 27, 2017 BY MIKAEL

Scraping, extracting and mapping geodata from Twitter

https://hegemonni.carto.com/viz/d01673a6-0f25-11e7-8671-0ef24382571b/embed_map

This post gives a brief and general introduction to scraping Twitter with Python, extracting geodata from this data and mapping it using Fusion Tables and CartoDB. I talked about this as well as some Facebook scraping I did with R in the working group on the Digitalization of Societies and Methods at the Annual Westermarck Conference in Tampere on Thursday.

Twitter is a tricky platform for any programmer who wants to extract geodata. Scrape tweets from any given hashtag and you'll only get a marginal amount of actual, pinpointed geodata with a longitude and a latitude. This has at least my experience when scraping Twitter.

What is one to do then, if what you are looking for is geodata? I came across this problem when I was scraping a hashtag for an international day of action against the private equity (or vulture) fund Blackstone. The events of the day were organized by a coalition of housing groups across several countries and continents and had an appointed hashtag, #StopBlackstone. I was following the action on the ground in Barcelona, where I was doing fieldwork with the compañerxs of la PAH for the project Learning in Productive Social Movements.

La PAH is a huge movement and it was clear from the outset that they would come up with a big mobilization. But what would the day look like in other places? I didn't see participant numbers from different places reported afterwards, but on the could deduce from pictures that most actions were small. But how about other types of engagement? How did the designated hashtag take off? Without falling into the old hyperbole of 'Twitter revolutions' I think it's still safe to say that social media engagement is a real part of how many movements build power. Yes, it is still only one metric among others when measuring the power of a movement, but it is still a metric. As such it can at least tell us *something,* even if it doesn't tell us nearly *everything*.

What I wanted to find out was what amount of unique users were engaging with the hashtag #StopBlackstone in different places. On the one hand, this is admittedly a blunt approach, which captures only a fraction of the properties of the network the formed around the hashtag. On the other hand, a map showing all the unique users that engaged with that hashtag would probably (and in the end did) produce a good approximation of the impact the action had in different places.

But where to start? For each tweet you send, Twitter includes two types of location data in the data that makes up the tweet:

1. Exact location data for the tweet.
2. General location data for the user.

Most people have turned off sharing of geodata for individual tweets. This means that there is no information on where they tweeted that exact tweet from. For instance, in the batch of 8310 tweets I scraped during one day of activities on the #StopBlackstone hashtag, only two users included the coordinates that the tweet was sent from. Five included the place they were tweeting from. Not much data to work with!

What most users do provide though, is information on their own general location. For the purposes of my research this was enough: it wasn't really important to know where one individual was tweeting from but to get an approximation of the geographical distribution of people tweeting.

What I did then, was a script to loop through the diverse geodata in each scraped tweet. I would first check if the tweet was provided with exact data on where it was sent from. If not, I would move on to the general location data provided by the user and saving that instead. The resulting list of JSON objects included locations for **709 users out of 1245**, who had produced **5184 out of the total 8310 of tweets** I scraped during that day.

To geocode and map this data I tried two different approaches. First, I passed the data through Google Refine to convert it from JSON to CSV. Then I uploaded it to Google Fusion Tables to geocode the data and produce this map:

https://fusiontables.google.com/embedviz?q=select+col3+from+1a0pv7jnJu7igDTCiSWDY

Because Fusion Tables has very few features for visualizing data points, I used CartoDB in my second approach to produce something more informative and pretty. This time I used OpenRefine to both convert the data to CSV *and* geocode it. Then the data was ready for CartoDB, that I used to produce the map at the beginning of this post. This map doesn't only show the geographical distribution of users, but also the amount of tweets (and retweets) that each user produced during the day as the size of the data point. In both maps you can find out more about a particular user by clicking on a data point.

On the ground, the #StopBlackstone action had at least 200 participants in Barcelona, with far smaller turnouts for the demonstrations in Dublin, Atlanta, Chicago, New York, Seattle, London and Japan. This is clearly reflected in the data on the map, where most locations match with actual places. There are also many outliers, of which most seem to be due to the elusive locations some people have in their Twitter description ("Peace is a continuous act", "Donde haya injusticia", "el mundo", "Union Sovietica", Nación Humana Universal" etc.).

In this post I will walk through the different steps of scraping tweets, extracting the relevant location data from them, geocoding this data and then mapping it. Not because this would necessarily be the best approach to my chosen problem, but because this was something I did in 2015 when I had just started working with Python, the Twitter API and geocoding, and I learnt a lot while doing it.

1. Scraping Twitter with Python

Twitter is the chosen medium to experiment on for a lot people who want to learn data science. The API of the platform is well documented and clear. Most programming languages have good libraries for interfacing with it. And, finally, Twitter data is much more straightforward to process than, for instance, Facebook data. That's also why I started working with Twitter data in 2015.

To get my data, I used the following elegant solution based on the Python Twitter library Tweepy and a script by Marco Bonzanini, with some added code for saving the tweets into a JSON file:

```
import tweepy
from tweepy import Stream
from tweepy.stream import StreamListener
from tweepy import OAuthHandler
import json

consumer_key = "YOUR CONSUMER KEY"
consumer_secret = "YOUR CONSUMER SECRET"
access_token = "YOUR ACCESS TOKEN"
access_secret = "YOUR ACCESS SECRET"

auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_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('FILENAME.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=['#HASHTAG_TO_SEARCH'])
```

If you install the Tweepy library and run this code in Terminal with an active hashtag, tweets will start pouring into your JSON file. Twitter imposes some limits on how many tweets you can access per second, but that only poses a problem for very active hashtags. These can produce several megabytes of data in less than a minute. To interrupt the code, just press ctrl+C.

2. Getting location data out of tweets

In the #StopBlackstone data I scraped with the code above in October 2015, the first scraped tweet is my own retweet of this tweet:

Tomorrow at 1pm in #NYC | Our Homes are NOT Commodities! #StopBlackstone https://t.co/3foWJ6XsE4 pic.twitter.com/BYBmLIQ8WG

— Occupy Wall Street (@OccupyWallStNYC) October 13, 2015

Each tweet I scraped is saved as a JSON object and includes massive amounts of data. I've made an online copy of the retweet of this tweet as it was when I scraped JSON object above here.

Out of the multitude of information stored in the tweet, I want to access location data for the tweet itself, and if that isn't available, the user. I wrote the following script to do this:

```python
import json

# Tweets are stored in in file "fname". In the file used for this script,
# each tweet was stored on one line

fname = 'Blackstone_STREAM.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'][tweet['coordinates'].keys()[1]][1]) + ", " + str(tweet['coordinates'][tweet['coordinates'].keys()[1]][0])
                    user_data["features"]["geo_type"] = "Tweet coordinates"
                elif tweet['place']:
                    user_data["features"]["primary_geo"] = tweet['place']['full_name'] + ", " + tweet['place']['country']
                    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 tweets.
                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 geo data, including 'location'"
    print "The users with geo data tweeted " + str(geo_tweets) + " out of the total " + str(total_tweets) + " of tweets."

# Save data to JSON file
with open('Blackstone_users_geo.json', 'w') as fout:
    fout.write(json.dumps(users_with_geodata, indent=4))
```

Running the script in Terminal produces the file "Blackstone_users_geo.json" along with the following dump:

```
The file included 1245 unique users who tweeted with or without geo data.
The file included 709 unique users who tweeted with geo data, including 'location'.
The users with geo data tweeted 5184 out of the total 8310 of tweets.
```

What the script actually does, is this: It iterates over all the tweets I scraped, extracting the user who authored the tweet or retweet. For each user, the script checks if they provided specific geodata on where they tweeted from. The vast majority didn't. Then the script moves on to location data that users provide voluntarily in their profile description. In most cases this includes some actual information on where this person is based, thus giving a clue on how engagement with the hashtag was distributed geographically.

The scripts creates a variable, "primary_geo", which saves the best possible geo data found. In addition, there are loops for figuring out how many tweets people made in total and how many users provided some kind of geodata.

3. Converting JSON to CSV with OpenRefine

Before geocoding and mapping the data, I needed to change the format of the data. Twitter serves tweets as JSON objects and this is also how I process them in my script. But neither CartoDB or Fusion Tables work well with JSON.

Instead of writing a script for converting JSON to CSV (or doing it in the original script) I just passed the data to OpenRefine, converted it to a table and downloaded it as a CSV file.

4a. Mapping geocoded data with Google Fusion Tables

Next I wanted to geocode this data, so that I can plot it on a map. The purpose is, in other words, to transform location data into a longitude and latitude marking it's position on planet Earth.

I usually use CartoDB out of personal preference, but due to their data limits I initially found Google Fusion Tables to be a better fit. To my great surprise, I noticed that Fusion Tables automatically geocodes any column in a table that is designated as 'location'. So, after uploading my data to Fusion Tables, designating my "primary geo" column as a "location" column and creating a map the Fusion Table map earlier in the post was born.

4b. Geocoding with Google Refine, mapping with CartoDB

I give Fusion Tables points for simplicity and clarity. But if you want to customize your map, it's not the best possible service. I quickly got frustrated when I wanted to make the data points reflect the amount of tweets each user had created. For this, CartoDB proved much better.

The problem with CartoDB is, in my experience, that their geocoding quotas are pretty strict. This means that you can only get a limited amount of data geocoded through their service, much less than the 709 places I needed to geocode.

So before uploading my data to CartoDB, I needed to geocode it somewhere. For this I again used OpenRefine, following the excellent tutorial at the Opensas blog. There were small differences to the geocoding done by Fusion Tables. My own location at the time was “Helsinki | Barcelona”. OpenRefine interpreted this as “Barcelona” and Fusion Tables as “Helsinki”. Both approaches use the Google Maps API, so I’m not sure why the difference. Anyway, the resulting map can be seen at the beginning of the post. I used the CartoDB web app to do small edits in the style of the map and borrowed a basemap from Mapbox.

Conclusions

Producing the maps for this semi-tutorial gave me some interesting information both on geodata on Twitter as well as the relationship between local activist networks and actions by international coalitions.

Regarding geodata and Twitter, it seems like one can produce informative displays of the geographical distribution of different networks even if the available data is quite inexact. Many users provide their location with enough accuracy to geocode it and map it. This data is static rather than dynamic, as it doesn’t provide information on where people tweet, but in large amounts it provides a good overview of where a specific hashtag has gained traction.

Looking at the maps produced here, we notice that even though the action was supposedly international, Spanish Twitter really dominated the #StopBlackstone hashtag. Even many data points in the US are actually Spanish (like “Valencia” that both Fusion Tables and Google Refine coded into Florida, even though it’s an autonomous province in Spain) and many random locations that seem to be in Spain, were coded into Latin America. The correct way to use the map would then to take a first look to get an approximation of were users were clustered. After that, you can click at the data points in a cluster to get a better understanding of why users were clustered there and assess the validity of the geodata.

Regarding the relationships between different levels of activity in networked movements: Combined with the fact that most pictures from the actions indicate a much larger mobilization at the Spanish offices of the Blackstone subsidiary Anticipa in Barcelona than anywhere else, it seems like the Spanish companerxs really dominated the action. My own gut feeling about this is that international actions usually only work if they are nurtured and amplified by networks that are more present in the everyday lives of people. La PAH has built its power on countless assemblies, stopped evictions and occupations of banks and this is what they brought to the international mobilization.

CODE

2 Replies to "Scraping, extracting and mapping geodata from Twitter"

Margarida Madaleno

OCTOBER 12, 2017 AT 5:17 PM

Hi,

Hope this message finds you well. Thank you for posting this, it's fantastic.

I'm a researcher working on scraping geo-tagged Tweets. I'm only interested in the first category of geocoding that you suggest (i.e. precise coordinates). My question is, why didn't you just do a bounding box within the stream.filter command? (See here: https://github.com/Ccantey/GeoSearch-Tweepy/blob/master/GeoTweepy.py). I'm asking because I wonder whether the bounding box only provides Tweets of type 1 (exact coordinates), or whether it also includes type 2 and 3 Tweets.

If the latter is true, I was thinking of running a script similar to the one you describe above, except using the bounding box as an initial search filter (instead of the hashtag you used), and then excluding your second and third iteration steps.

Thanks in advance!

Pingback: Pyhon3.4 TypeError: 'dict_keys' object does not support indexing | Mobilapka.cloud