Data From Twitter

Zachary C. Steinert-Threlkeld

09.08.2020

```
knitr::opts_chunk$set(echo = TRUE, eval=FALSE)
knitr::opts_knit$set(root.dir ='/Users/Zack/Documents/UCLA/Courses/EventDataFromSocialMedia/')
knitr::opts_chunk$set(tidy.opts=list(width.cutoff=80),tidy=TRUE)

library(rtweet)
```

SET-UP

To connect to Twitter, you need a token. If you were able to create a token in the last class, use that. If not, use "rtweet}'s token.

This code chunk repeats the token creation process.

```
api_key <- "cv0c4OkCkUcfw6EdIvlX9B2ay"
api_key_secret <- "EW7um8Ag4SD6KGfmuQEwqcv2OJItNcwHJJWJYtKK7lSWzFJ84q"
access_token <- "use_your_own_please"
access_token_secret <- "use_your_own_please"

wildfire <- create_token(app = "Tracking_Wildfires", consumer_key = api_key, consumer_secret = api_key_s
    access_token = access_token, access_secret = access_token_secret)

wildfire  # Here is the token
```

The token `wildfire` is now saved as an environment variable. This chunk removes it and then shows you how to load it.

```
rm(wildfire)

wildfire <- get_token()  # get_tokens() if you have created several
```

Note that once you create a token, you do not need to run the below code again. You can just use the `get_tokens()` function like in the above code snippet.

If you were not able to create an application in the previous session, `rtweet` will create a user token for you via its app. You need the `httpuv` package installed. Then, when you first attempt to connect to an endpoint, R will open a tab in your browser and ask you to allow the `rstats2twitter` app to access your Twitter account. If you allow it, you will then have access to the Twitter API.

```
install.packages("httpuv")
library(httpuv)  # If you will use rtweet's token

# This code will open a browser tab asking you to authorize the rstats2twitter
# app.
rt <- search_tweets("#rstats", n = 18000, include_rts = FALSE)

get_token()  # rstats2twitter
```

If you did not create a token on your own, run the sample code below and authorize the `rstats2twitter` app. This approach only works if you have a Twitter accoutn, and the token is only valid via an interactive

seesion. I therefore do not recommend it if you will run a script, which becomes more common as the time intensiveness of your task grows.

STREAMING API

Now, let's start collecting tweets. You will notice that I pass my token. This value is NULL by default because it reads your R environment, so it will load `rstats2twitter` if you do not have your own token. You could pass `token=rstats2twitter` if want, though it is not necessary.

Technically, we are connecting to the `POST statuses/filter` endpoint. ## RANDOM SAMPLE By default, `stream_tweets} connects for 30 seconds using the token in your environment and returns the matching tweets as a data frame. Without specifying values toq`", a 1% random sample of Twitter is returned.

```
s1 <- stream_tweets()
```

Occassionally, you may get a parsing error message. That message means the tweet contains a typo as delivered, corrupting the entire data frame. The below screenshot shows what can happen.

```
> s1 <- stream_tweets()
Streaming tweets for 30 seconds...
Finished streaming tweets!
Error: parse error: after array element, I expect ',' or ']'
          :"1598983470665"},785068032","user_id":1205106198271401987,"
                     (right here) ------^
> s1
Error: object 's1' not found
```

This error provides a nice segue into learning arguments for `stream_tweets` that are not related to Twitter.

```
# Longer connection
s2 <- stream_tweets(q = "", timeout = 60, file_name = "Data/random_60s")  # Units are seconds

# Do not convert the returned results to a data frame.
s3 <- stream_tweets(q = "", timeout = 10, parse = FALSE, file_name = "Data/random_10s")
```

If you try to access `s3`, you will see it is not easy. Solve that with `parse_stream` or by modifying the query to Twitter. Note that if there is a formatting error like for `s1`, `parse_stream` will still error out.

```
s3b <- parse_stream(path = "Data/random_10s.json")
s3b <- data.frame(s3b)  # I hate tibbles

# Now the tweets are in a format we are used to
dim(s3b)
head(s3b)
sort(names(s3b))

# Or specificy a file name when you stream.
s4 <- stream_tweets(q = "", timeout = 10, parse = FALSE, file_name = "Data/random_sample")  # .json by

# This one will be used for teaching the next session
s5 <- stream_tweets(q = "", timeout = 60 * 10, parse = FALSE, file_name = "Data/teaching_random_sample_
```

FILTERS

While it is fun to collect a random sample of all tweets, finding tweets within that sample relevant to event data is going to be very hard. Most people do not talk about politics, so 1% of "most people" is going to be few tweets. Instead, it will be more productive to consider more focused approaches to collecting tweets. We will now learn how to do that.

Twitter documents these parameters here.

Keywords

Perhaps there are specific types of events in which you are interested, in which case there are probably keywords more likely to appear in tweets containing those events. Twitter allows up to 400 keywords per connection. You can use keywords to find tweets with URLs, though matching is at the domain level, i.e. you cannot pass "http" or "www" to find tweets with any URL.

Commas are ORs, spaces between words are AND.

Chinese, Japanese, and Korean are not supported.

If you want to know when a user is mentioned, pass their screen name as a keyword.

```
key_protest <- "protest,riot,demonstration"
protest <- stream_tweets(q = key_protest, timeout = 30, parse = FALSE, file_name = "Data/tweets_protest

key_conflict <- "clash,war,attack,fight,assault"
conflict <- stream_tweets(q = key_conflict, timeout = 30, parse = FALSE, file_name = "Data/tweets_confli

# This one will be used for teaching the next session.
key_mixture <- paste(key_protest, "this", "nba", "nfl", "hollywood", "cars", "boats",
    "sky", "weather", "police", "nature", "boat", "the", sep = ",")
mixture <- stream_tweets(q = key_mixture, timeout = 60 * 10, parse = FALSE, file_name = "Data/teaching_
```

Users

Following users is a useful way to follow news organizations, politicians, and institutions (the military or international organizations, for example). Their tweets can then later be filtered to identify events. If you are following the CAMEO coding scheme, then a tweet could be considered a public statement. Depending on its content, a tweet could represent actions such as `appeals`, `cooperation`, `consultations`, `investigating`, `demanding`, `disapproving`, `rejecting`, or `threatening`. Whether a tweet itself counts as event or a collection of them documents an actual offline action is outside the scope of today's class.

Twitter also allows you to follow up to 5,000 public users at once, using their user ID. Twitter will then provide tweets from those users, including retweets, replies to a user's tweet, or retweet's of a user's tweet. It does not provide mentions.

First, however, you need a user's ID. There are services that will do that for you, but we can also use `rtweet`.

```
news <- c("nytimes", "BBCMonitoring", "TimesLIVE", "ElNacionalWeb")
users <- lookup_users(users = news)
users_ids <- users$user_id
users_ids <- paste(users_ids, collapse = ",")
```

Now, we can pass `users_ids` to `stream_tweets`.

```
news <- stream_tweets(q = users_ids, timeout = 20, parse = FALSE, file_name = "Data/tweets_news")
```

Locations

My favorite way of accessing the streaming API is to request only tweets where Twitter has assigned a location. Approximately 2% of tweets contain a location (users must opt in per tweet) and Twitter's complicated sampling actually means we get about half of these tweets, i.e. .01/.02, not .01*.02. (In my experience, however, about half of these geolocated tweets only contain country information.) To access tweets with location, you pass a bounding box of the southwest corner and northeast corner where each corner is given as longitude then latitude.

Determining a bounding box is tricky. I have always used bboxfinder.com, but `rtweet` has a function, `lookup_coords` that interfaces with Google Maps. To use the function, you need a Google Maps API key unless your bounding box is an American city, the USA, or world, in which case it is hardcoded into `rtweet`.

```
usa <- stream_tweets(q = lookup_coords("usa"), timeout = 20, parse = FALSE, file_name = "Data/tweets_usa
world <- stream_tweets(q = lookup_coords("world"), timeout = 20, parse = FALSE, file_name = "Data/tweets

bbox_world <- lookup_coords("world")
world <- stream_tweets(q = bbox_world$box, timeout = 20, parse = FALSE, file_name = "Data/tweets_world2
```

Here is an example using coordinates we provide. Note that because countries are never actually rectangles, you will receive some tweets from outside the country you want to bound. You will have to remove those tweets on your own using the tweet's `{place}{country_code}` field.

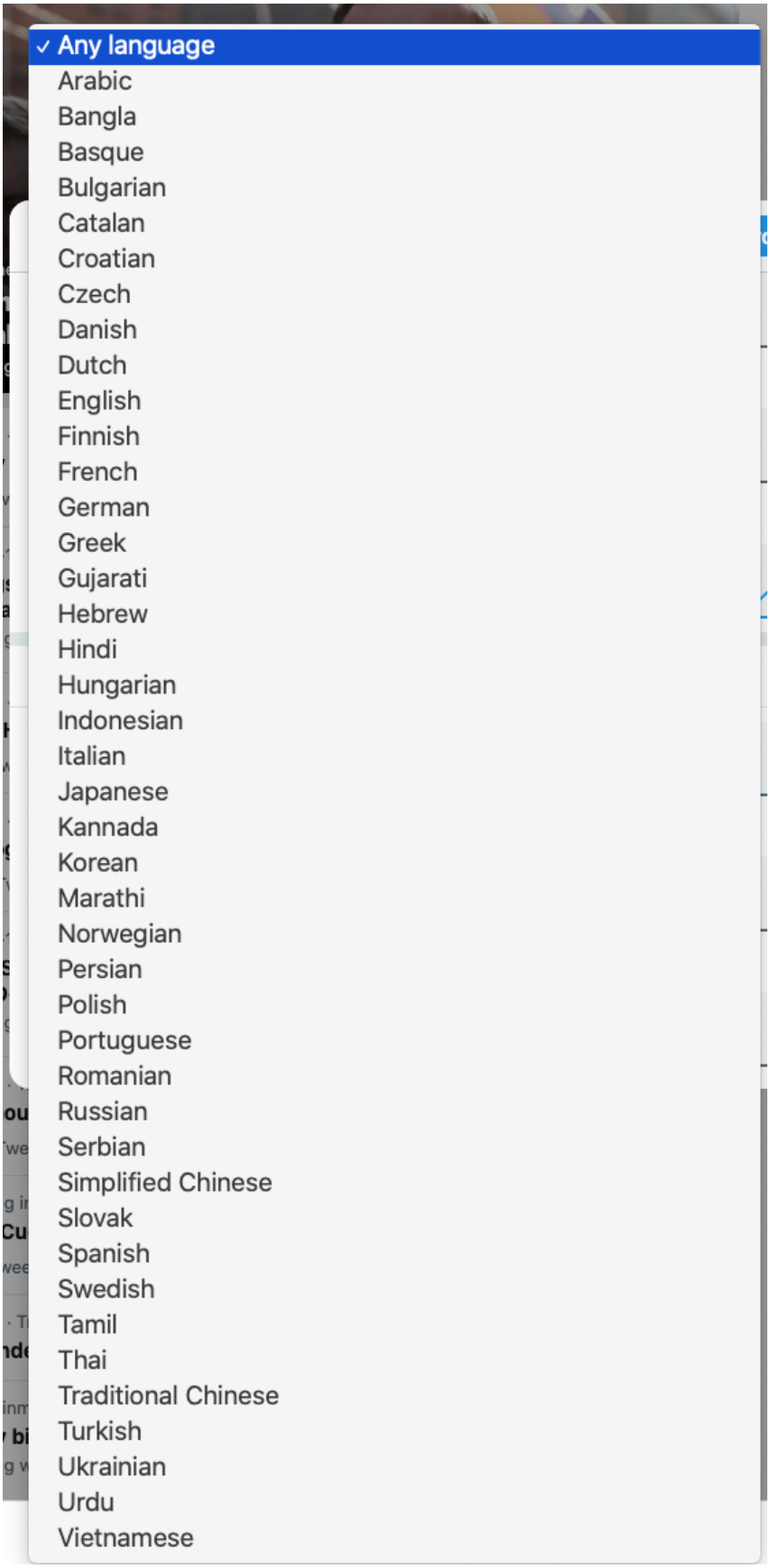
```
bbox_turkey <- c(25.795898, 35.764343, 45, 42.179688)
turkey <- stream_tweets(q = bbox_turkey, timeout = 20, parse = FALSE, file_name = "Data/tweets_turkey")
```

Retweets do not contain location but replies do. For event data, this is a benefit.

Bounding boxes are treated as OR, not AND.

Languages

Finally, language is the last parameter you can pass to the streaming API. Twitter recognizes the following languages:


✓ Any language
Arabic
Bangla
Basque
Bulgarian
Catalan
Croatian
Czech
Danish
Dutch
English
Finnish
French
German
Greek
Gujarati
Hebrew
Hindi
Hungarian
Indonesian
Italian
Japanese
Kannada
Korean
Marathi
Norwegian
Persian
Polish
Portuguese
Romanian
Russian
Serbian
Simplified Chinese
Slovak
Spanish
Swedish
Tamil
Thai
Traditional Chinese
Turkish
Ukrainian
Urdu
Vietnamese

The two-letter language codes are available here (or anywhere "BCP 47" codes are available).

```
eng <- stream_tweets(q = "en", timeout = 20, parse = FALSE, file_name = "Data/tweets_english")
dutch <- stream_tweets(q = "nl", timeout = 30, parse = FALSE, file_name = "Data/tweets_dutch")
```

Off the top of my head, the largest missing language is Tagalog. Sometimes it is classified as unknown ("und"), other times English.

As recently as May 2018, Twitter treated language as an AND filter, meaning you had to supply keywords to then filter by language. No longer having to supply keywords particular to a lanugage saves a lot of programmer time.

REST API

Now, let us turn to the REST API. REST stands for Representational State Transfer, and it is widely used across the web; it is not unique to Twitter. It does not refer to any particular endpoint but rather to all of those that do not *new* information.

For the purposes of event data, Twitter's REST API only has two end points that are useful, the `Search` endpoint and `GET statuses/user_timeline`. They each have limitations, discussed below.

Search

The free Search endpoint, `GET search/tweets` only provides tweets from the past ~7 days, according to Twitter. Here is the Twitter documentation for this endpoint.

Results returned are are also "focused on relevance *and not completeness* [emphasis original]".

Queries can only contain up to 500 characters, which Twitter recommends limiting to 10 keywords and operators.

Location requests are given as a latitude, longitude, and radius, not a bounding box. Twitter will infer location from a user's self-reported location, which it does not do for the Streaming API, so this capability may be advantageous.

Per 15 minutes, you are allowed 450 requests that can return up to 100 tweets. This equals a maximum of 4,320,000 tweets per day, about 20% fewer than using the Streaming API.

Given these limitations, I only consider using the search endpoint at the start of a project if I need tweets from the last seven days. For example, if I want to follow a new protest event, I will connect to the Streaming API and follow keywords needed and work with the Search API to find those keywords in the previous seven days.

Finally, note that you can pay for enhanced access to the Search API. Twitter calls these endpoints "Premium search" and "Enterprise search", the difference being how much data is returned and therefore how much each costs. To see pricing, you will have to go to your developer homepage. The Premium pricing is $99 to $1,899 per month, depending on how many tweets you want.

```
kenosha <- search_tweets(q = "kenosha", n = 200, parse = TRUE)
table(kenosha$is_retweet)

kenosha_nort <- search_tweets(q = "kenosha -filter:retweets", n = 200)
table(kenosha_nort$is_retweet)

kenosha_media <- search_tweets(q = "kenosha filter:media", n = 200)  # image or video.  Could use filte
table(kenosha_media$media_url)  # Unclear what filter:media does
```

```
protests <- search_tweets(q = "protest,demonstration,march", n = 200)
protests2 <- search_tweets(q = "protest,demonstration,march filter:media -filter:retweets",
    n = 200)

kenosha_location <- search_tweets(q = "", geocode = "42.588081,-87.822899,10mi",
    n = 200)
kenosha_protest_location <- search_tweets(q = "protest", geocode = "42.588081,-87.822899,10mi",
    n = 200)
```

Note that rate limits may impede if you request a large number of tweets. Pass `retryonratelimit=TRUE` as an argument in that situation.

User Tweets

To go back further in time, you can query a user's 3,200 most recent tweets using `GET statuses/user_timeline`. For users with fewer than 3,200, this is their entire tweet history. For users with more, it could be a week's, month's, or year's worth of data; you will not know until downloading. You could use this endpoint to collect a user's tweets in real time, but it is simpler to just follow the user in real time via `POST statuses/filter`.

Note that Twitter accepts user names or user IDs here. It is best to use the ID because it does not change while the name can. For example, if I change my screen name from ZacharyST to ZacharyST2 and you try to download ZacharyST's tweets, you will receive 0 tweets; if you use the user ID of ZacharyST, you will retrieve my tweets.

```
ids <- strsplit(users_ids, ",")[[1]]  # We will use the user IDs of news organizations from earlier.
one <- get_timeline(ids[1], n = 3200)  # Could set parse=FALSE and write to file

dim(one)
one$created_at[1]  # newest tweet
one$created_at[3200]  # oldest tweet

all <- get_timeline(ids, n = 3200)
nrow(all) == 3200 * length(ids)
table(all$screen_name)
```

NOTES

"Stream", "REST"

When discussing Twitter data, you will often see people refer to the "Streaming API" and "REST API". These are old terms Twitter used but no longer does, but they have stuck. "Streaming API" refers to tweets from the `POST statuses/filter` endpoint.

Rate Limits

Twitter, like all services of which I am aware, limits how much data they will return in a time window. It is your responsibility to adhere to these limits. Adhering to them makes code more complicated but is necessary. Rate limits essentially only apply to the REST API. If you are disconnected from the streaming API, make sure your script does not try reconnecting too often; if it does, Twitter will return a 429 error message (too many requests).

Some functions in `rtweet` will automatically handle rate limits for you.

Here are Twitter's rate limits. Note that some endpoints also impose a per-day limit. The only one I have used that has this restriction is `GET statuses/user_timeline`, which only allows 100,000 requests per day.

Streaming Connection Time

It is a good idea to keep your connectiosn to the Streaming API relatively short. The files get large quickly. A disconnection early in a connection also means you will miss a lot of subsequent data.

I connect once every hour. I use `cron` to launch my script every hour, and the script connects for 3,600 seconds (60*60 seconds). These files are approximately 500 megabytes per hour, though I do not pass any filters that could result in smaller files. I do not use larger windows, like 4 or 24 hours, because a disconnection at hour 1 would cause me to miss 3 or 21 hours' of data. You could build your code to reconnect on a disconnect.

JSON vs. Rectangles

I prefer to save the raw tweets directly to file and later load them. There are two advantages of this approach. First, asking `rtweet` to parse puts a greater burden on your computer while it is trying to ingest a lot of tweets. If connecting to the Streaming API, this could cause you to miss some or wait a long time for the data frame to be created. Second, the contents of tweets often change. Since I started collecting data, Twitter has added new media types, introduced entities, and will soon add annotations (topics) to tweets. `rtweet` may handle these expansions, but you are reliant on the creator either having coded `parse_tweets` to work that way or updating the package quickly in reponse to changes. If this does not happen, you will lose these data. If you store the raw tweets, however, you do not lose the data, and you can later decide whether to keep them or not. Third, many of the tweet fields, such as `favourites_count` or `url` (profile link) are probably not relevant. Putting them in a data frame therefore takes RAM that would be better kept free. Reading in the raw tweets yourself allows you to decide which columns matter.

File Names

If you pass `file_name=NULL` to `parse_tweets`, the file created will include a timestamp ending in YYYYMMDDHHMMSS. This feature is nice because it means you will get unique file names for new connections. If you want a custom file name, you will overwrite any tweets previously using that name unless `parse_tweets2(append=TRUE)`. Notice the slightly different function name.

In my work, I pass a custom file name because each connection lasts for one hour, so I want the filename to include the starting hour.

YOUR TURN

Keywords

Create a string of keywords and download them.

```
Go!
```

Location

Download tweets from Dallas, Texas.

```
Go!
```

Download tweets from Washington, D.C..

```
Go!
```

Language

Download tweets in German.

```
Go!
```

Download tweets in German or from Germany.

```
Go!
```