Access Twitter posts by country

Jerid Francom

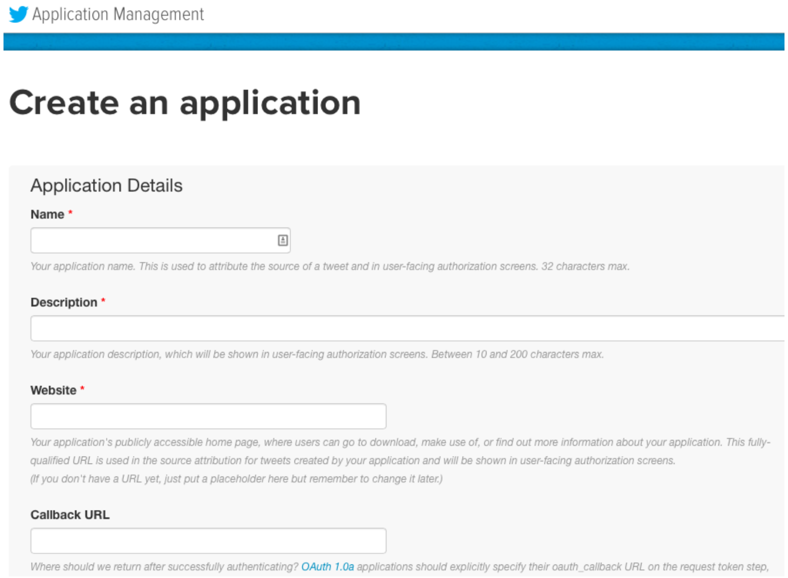
December 14, 2014

In this ExploRation I will cover how to retrieve and filter tweets from Twitter by country. The first step will be to create and connect to the Twitter API using the `twitteR` and `ROAuth` packages. If you don't already have one you will also need to register for a Twitter developer account and then create an application. This will give you access to an API key and secret. With these packages and credentials, we then will use `streamR` to download tweets. After retrieving the data, we will then proceed to filter those tweets that fall within the borders of a certain country with the `sp` package.

Some other packages you will need include: `plyr`, `data.table`

Twitter API authenication

Before we get started on the R side, we'll need to set up a Twitter application. First, log in to your Twitter developer account (or create one). Then you'll follow the link 'Manage Your Apps' and select 'Create New App'.



Create an application

Application Details

Name *

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max.

Description *

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your application. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens.
(If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step,

Fill out the form and then scroll down and accept the Developer Agreement. Then you will be able to select your app's name and grab your `API Key` and `API Secret` under the 'Keys and Access Tokens' tab. Add these credentials, the request, access, and authorize urls, and a path to a file to store the resulting authentication information. The `oauth.file` object is not strictly necessary, but it means that you will not need to perform the upcoming API handshake every time you want to interface with Twitter.

```
api.key <- my.api.key # your consumer key
api.secret <- my.api.secret # your consumer secret
request.url <- "https://api.twitter.com/oauth/request_token"
access.url <- "https://api.twitter.com/oauth/access_token"
authorize.url <- "https://api.twitter.com/oauth/authorize"
oauth.file <- "myoauth.RData"
```

Now that we have this key information set up, it's on to the authentication. Load the RCurl and ROAuth packages. Then we set some RCurl options to help us create the my.oauth data. OAuthFactory creates a new set of authentication parameters based on our credentials and then we perform the 'handshake'. Running my.oauth$handshake() will access your developer account at which point you will be asked to give permissions to this application. After accepting, copy the PIN and paste it into the prompt in R. If you want to store this object, go ahead and save it to your hard disk.

```r
library(ROAuth)
options(RCurlOptions = list(capath = system.file("CurlSSL", "cacert.pem",
                                                package = "RCurl"),
                            ssl.verifypeer = TRUE))
my.oauth <- OAuthFactory$new(consumerKey = api.key,
                             consumerSecret = api.secret,
                             requestURL = request.url,
                             accessURL= access.url,
                             authURL = authorize.url)
my.oauth$handshake()
save(my.oauth, file = oauth.file)
```

From now on we can just load the myoauth.RData data and confirm that everything is alright.

```r
library(twitteR)
load(file = "myoauth.RData")
registerTwitterOAuth(my.oauth) # check status
```

```
## [1] TRUE
```

Get tweets

And on to getting some data! streamR::filterStream() gives us access to Twitter streaming data.(Use streamR::userStream to get specific user timelines.) The parameters of this function may need some explaning: here file is set to "" to redirect the stream to the console, locations is set to cover all geo-coordinates, which has the effect of only retrieving tweets with coordinate information, timeout is the time we want to hold the stream window open, and oauth is where our credentials vouch for our application.

```r
library(streamR)
world.tweets <- filterStream(file="", # redirect to the console
                             locations = c(-180,-90,180,90), # geo-tweets
                             timeout = 60, # open stream for '60' secs
                             oauth = my.oauth) # use my credentials
```

The result assigned to world.tweets is a JSON string. To parse this data into a more user-friendly tabular format, we use streamR::parseTweets().

```r
world.tweets <- parseTweets(world.tweets)
```

After parsing the tweets we end up with 42 pieces of meta data for each including:

text, retweet_count, favorited, truncated, id_str, in_reply_to_screen_name, source, retweeted, created_at, in_reply_to_status_id_str, in_reply_to_user_id_str, lang, listed_count, verified, location, user_id_str, description, geo_enabled, user_created_at, statuses_count, `followers_count, favourites_count, protected, user_url, name, time_zone, user_lang, utc_offset, friends_count, screen_name, country_code, country, place_type, full_name, place_name, place_id, place_lat, place_lon, lat, lon, expanded_url, url`

For our current purposes much of this information is not necessary. Let's focus on only a few key columns: language, latitude, longitude, and text.

```
world.tweets <- world.tweets[, c("lang", "lat", "lon", "text")]
```

At this point you might want to write this data to disk for future access. One issue that I have found when working with text from tweets is that there are various characters that end up causing problems for reading the data back into R. In particular carriage returns end up in some of the text and misalign the columns. So, before writing the data we'll remove any `\\n+` in the tweet `text` field. And for whatever reason some tweets come with incorrect/ illegal `lat`, `lon` coordinates. Let's filter them too.

```
# Remove extra line breaks
world.tweets$text <- gsub("\\n+", "", world.tweets$text)
# Remove spurious coordinates
world.tweets <- subset(world.tweets,
                       (lat <= 90  & lon <= 180  & lat >= -90 & lon >= -180))
# Write data to disk
write.table(x = world.tweets, file = "worldtweets.tsv",
            sep = ",", row.names = FALSE, fileEncoding = "utf8",
            quote = TRUE, na = "NA")
# Clean up workspace
rm(list = ls())
```

To read in data I use `data.table::fread()`. It's a screaming fast import function for tabular data. It creates a `data.table` structure which is also a great alternative to the `data.frame`.

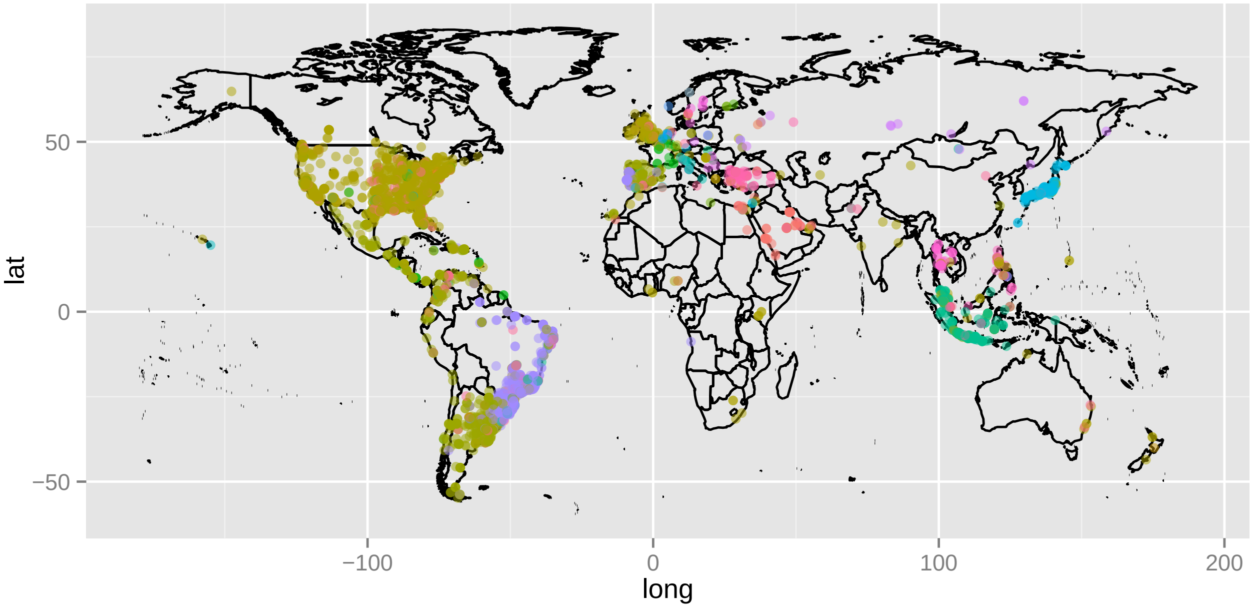
```
library(data.table)
world.tweets <- fread(input = "worldtweets.tsv", sep = ",",
                      header = TRUE, data.table = FALSE)
```

Clipping geo-coordinates

Before we get to filtering the tweets by country. Let's take a look at where the data we've captured originates from.

```
library(ggplot2)
world.map <- map_data(map = "world") # get the world map
world.map <- subset(world.map, subset = region != "Antarctica") # remove this region
p <- ggplot(world.map, aes(x = long, y = lat, group = group)) +
  geom_path() # base plot

p + geom_point(data = world.tweets, # plot tweet origin points
               aes(x = lon, y = lat, color = lang, group = 1),
               alpha = 1/2) + theme(legend.position = "none")
```



I've added color to the plot to indicate the languages (according to Twitter) that are in the data.

In order to filter these tweets from around the world by country we need to get spatial polygon data for the country(ies) of interest. In this example, we'll use the US data found on the GADM website. Various formats are available, but for our purposes we'll take the convenient route and select the `.RData` file. You will be presented with various level files which correspond to rough to fine-grained detail. We'll select the Level 0 data.

```
# Download SpatialPolygonsDataFrame in .RData format
url <- "http://biogeo.ucdavis.edu/data/gadm2/R/USA_adm0.RData"
file <- basename(url) # gets the file's name
if (!file.exists(file)) { # If the `file` hasn't been downloaded, do so now
  download.file(url, file)
}
load(file = file) # Now load the `file` from disk
```

The next step is to extract our tweet coordinates and convert them into a SpatialPoints object with the same projection as the `gadm` data that we downloaded. This ensures that we are going to compare apples-to-apples come time to filter by country.

```
coords <- world.tweets[, c("lon", "lat")] # extract/ reorder `lon/lat`
library(sp)
coordinates(coords) <- c("lon", "lat") # create a SpatialPoints object
proj4string(coords) <- proj4string(gadm) # add `gadm` projection to `coords`
```

Clipping the data that falls outside of the spatial polygon couldn't be easier: we just subset the original coordinates `coord` extracted from `world.tweets` by the `gadm` object. The result returns only those coordinates that fall within the spatial object –that is, within the USA.

```
system.time(usa.coords <- coords[gadm, ]) # filter tweets
```

```
##    user  system elapsed
##  46.043   0.495  46.684
```

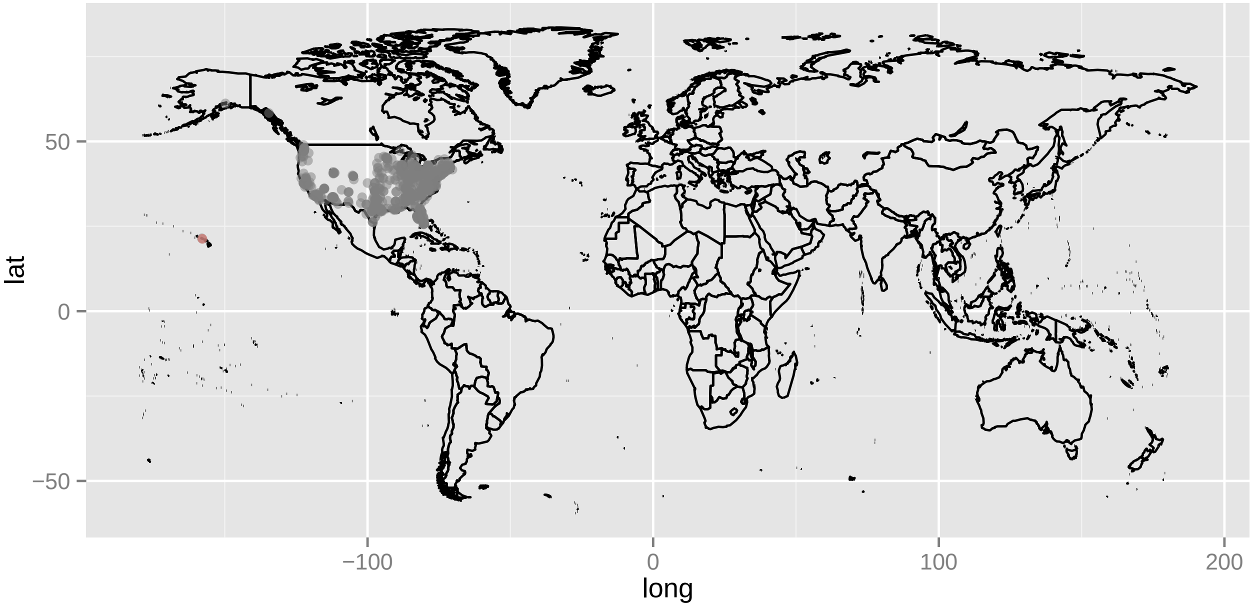
```
usa.tweets <- as.data.frame(usa.coords@coords) # extract coordinates
```

We'd like to attach these points to the relevant data from the `world.tweets` data.frame, and remove the rest. To do this we use `join()` from the `plyr` package. Since in both the `usa.tweets` and `world.tweets` data.frames the coordiantes are stored under the same column names (`lat`, `lon`) we don't have to specify what columns to join by –if not specified `join()` will join on matching columns.

```
library(plyr)
usa.tweets <- join(usa.tweets, world.tweets)
```

The result is a data.frame `usa.tweets` that contains the columns lon, lat, lang, text for tweets originating from the US. Let's visualize our work to make sure that we indeed have isolated the relevant tweets.

```
p + geom_point(data = usa.tweets, # plot tweet origin points
               aes(x = lon, y = lat, color = lang, group = 1),
               alpha = 1/2) + theme(legend.position = "none")
```



So there you go. We've downloaded Twitter posts via the official API, wrote that data to disk, read in the data from disk, and clipped coordinates not falling within the United States.