

Twitter Stream Project

Joel Willers

April 30, 2014

Objective

Twitter is a powerful social media tool used across the world to communicate. Simply using Twitter to send and receive messages, though, pales in scope to the information that monitoring the streaming feed can allow us to harness. Since most accounts on Twitter are public, we are able to get a 'pulse' on the current mood of the world by tying into the live Twitter stream, made available to us by the Stream API. This exercise shows just one application of the power that can be accessed online today. When a search term is entered, points are shown on the map of the world corresponding to the location of a person that just mentioned that term in a public Tweet.

Four packages were used in this application:

- streamR - Ties into the Twitter data stream and parsing the data.
- OAuth - Used for authorizing the transactions with Twitter.
- shiny - Allows us to make responsive displays and run the App
- maps - Provides a world map to display locations.

```
library(shiny)
library(maps)
library(OAuth)
library(streamR)
```

Methods

Data collection from the streamR package begins with authorization. Every person that uses this application has to have their own Twitter information to use. You will need to location your 'Consumer Key' and your 'Consumer Secret' from Twitter by going to <https://dev.twitter.com/> and creating an App. Do NOT share this information with anyone, as it will allow anyone to post directly to your account, along with any other Twitter functions.

```
## Commented out so it does not run requestURL <-
## 'https://api.twitter.com/oauth/request_token' accessURL <-
## 'https://api.twitter.com/oauth/access_token' authURL <-
## 'https://api.twitter.com/oauth/authorize' consumerKey <- 'YOURKEY'
## consumerSecret <- 'YOURSECRET' my_oauth <-
## OAuthFactory$new(consumerKey=consumerKey, consumerSecret=consumerSecret,
## requestURL=requestURL, accessURL=accessURL, authURL=authURL)
## my_oauth$handshake(cainfo = system.file('CurlSSL', 'cacert.pem', package =
## 'RCurl'))
```

After this is done, you have to enter a PIN given by Twitter at the prompt. Once this is done, you can save this authorization to a file, so that you will not have to do it again.

```
# Save the credentials (commented out since we already have them saved.)
# save(my_oauth, file='credentials.RData')

# Load the credentials
load(file = "credentials.RData")
```

ui.R

Like all shiny apps, this app uses ui.R and server.R. In this ui.R we have

```
# Define UI for Twitter content shinyUI(fluidPage( # Application title
# titlePanel('Choose a topic to follow on Twitter'), # Sidebar with control to
# type in a topic (by default we have a happy face) sidebarLayout( sidebarPanel(
# textInput('contentText', 'Enter search term:', value = ':)'),
# submitButton('updateData', 'Search'), helpText('This updates every 10 seconds,
# so please be patient.') ), Show a map of the World mainPanel(
# plotOutput('worldMap'), tableOutput('view') ) ) )
```

server.R

This section is what does most of the data collection and modification. Due to the nature of shiny apps, most of the code will be commented out, but in order for an example to run, some example code will also be included following a comment of 'example'.

```
# Construct a data frame using the filterStream function. This is a nice base to
# start from
tempTweets <- filterStream(file.name = "", locations = c(-180, -90, 180, 90), tweets = 10,
  oauth = my_oauth)
parsedList <- parseTweets(tempTweets)

## 6 tweets have been parsed.

# Initialize an empty data frame
datalist <- head(parsedList, 0)

## The main function for the map shinyServer(function(input, output, session) {
## Take the data frame and make a reactive counterpart values<-reactiveValues()
## values$datalist<-datalist

# Set a timer for 11 seconds to restart the search process autoTime <-
# reactiveTimer(11000, session)

# observe ({ Trigger the search process using the autoTime function created
# earlier autoTime() tempTweets <- filterStream( file.name='',
# track=input$contentText, timeout=9, oauth=my_oauth ) Example
tempTweets <- filterStream(file.name = "", track = ":"), timeout = 9, oauth = my_oauth)

# Because of the connection problems, we need to watch for error messages.
```

```

if (any(tempTweets == "Easy there, Turbo. Too many requests recently. Enhance your calm.")) {
  parsedList <- data.frame("Session error, please wait")
  Sys.sleep(3)
} else if ((length(tempTweets) > 0) & (any(tempTweets != "Exceeded connection limit for user")))) {
  parsedList <- parseTweets(tempTweets)
  parsedList <- subset(parsedList, !is.na(parsedList$place_lon))
  # values$datalist<-parsedList
}

## 514 tweets have been parsed.

# })

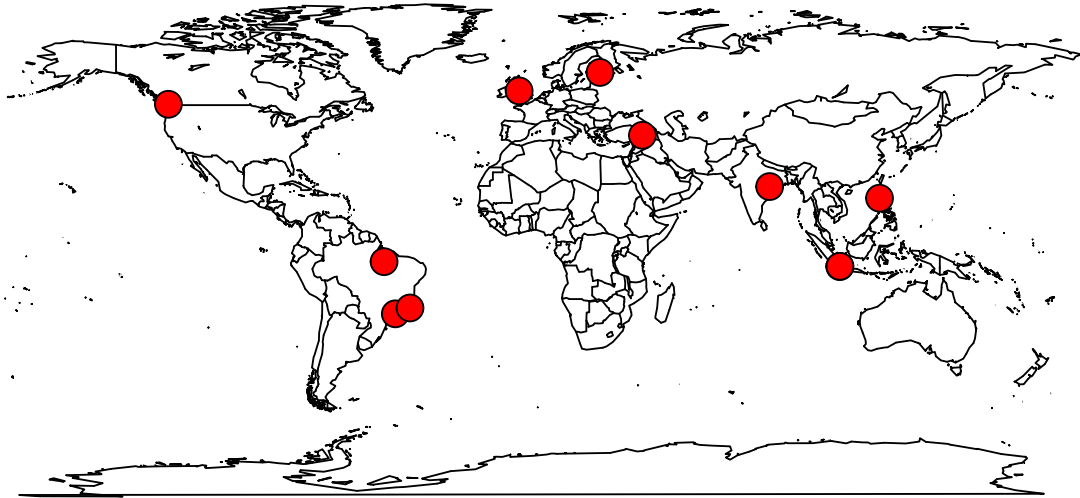
# Make the map output$worldMap <- renderPlot({

# map('world') Make sure to show only when we have data if
# (nrow(values$datalist)>0) { Make the points points(values$datalist$place_lon,
# values$datalist$place_lat, pch = 21, bg='red', cex=2) } }) Example
map("world") + points(parsedList$place_lon, parsedList$place_lat, pch = 21, bg = "red",
  cex = 2)

## Error: non-numeric argument to binary operator

# output$view <- renderTable({ if (nrow(values$datalist)>0) { values$datalist }
# else data.frame('No results yet. You can wait or enter a new search term.') })
# })

```



Future Goals

This application has quite a bit of future potential. Because of this, there are many future goals for the project.

- Reduce the amount of refresh time to 1 sec or less
- Add more search terms to the search with different colors
- Give users the ability to adjust the display properties of the dots on the map
- Record the sessions in order to replay the recordings
- Allow for other time-related data streams to be displayed using this map and a function

Obstacles overcome


This was an interesting project fraught with obstacles. I was looking forward to being able to add a great deal of functionality by utilizing existing packages. However, the streamR package does not access the Twitter stream in the way the API was designed. A user is supposed to tie into the stream and view the data as it comes through what is known as the 'firehouse'. The streamR package connects to the stream, gathers the information for a set amount of time, then releases the stream. This causes two main problems. First, nothing else can happen in the R program until the stream is released. Data is not accessible during this time, nor can any other functions progress. This does not happen in the background, but in the foreground while we wait. The second problem is that there is a limit to how often the stream can be accessed. It comes to about six times per minute. This solution was to use a ten second stream six times a minute, but even then, there are connectivity problems.

The stream was meant to stay open, not be repeatedly opened and closed for surgical use. There is a reason it is called the 'firehouse'. In the future, it could be possible to control an external script through the shiny app. When the search term is entered, the background program starts collecting data into a database. While the connection stays open, so does the database collecting information. The shiny app, then, accesses the database instead of the actual stream.

Another solution is to rewrite the streamR package to keep the stream open. I assume there were technical difficulties with this somehow, which is why it was abandoned. Still, it is a possible option to pursue.

Summary

While this is not final iteration of this application, it is a very solid first step. The stream is accessible, search terms are entered and data is displayed. There is still a long way to go, but with the technical limitations, this is about as far as can be done without writing external programs or new packages. That seems like a great summer project, though.

map.png