Doing Linguistic Analysis with Twitter Data: The Basics in R

Eric Wilbanks
wilbanks.ericw (at) gmail.com

North Carolina State University
Linguistics Brownbag

August 28, 2015

Intended Audience

Who are You?

- New to doing Twitter Analysis
- Interested in Dialectology, Lexical Variation, Sentiment Analysis, or Network Data

Who am I?

- NOT a Twitter Data Expert
- Sharing my limited experience to get you inspired to really dig deep and learn more on your own

What We'll Be Doing

We'll be briefly covering some theoretical and methodological considerations before working directly with the code in these slides and the data you should be have access to with these slides.

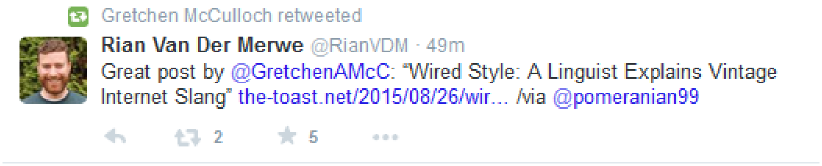
Outline

Introduction and Motivation

Why Do We Care About Twitter?

1. Enormous quantities of linguistic data out there for the taking.
2. 304 million active users (`http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/`)
3. Each day an average of 500 million tweets are sent out (`http://www.internetlivestats.com/twitter-statistics/`)
4. Learning to work with Big Data improves your problem-solving abilities and is a different way of approaching variation.
5. Improve your computational and scripting skills!

Anatomy of a Tweet


Gretchen McCulloch retweeted
Rian Van Der Merwe @RianVDM · 49m
Great post by @GretchenAMcC: "Wired Style: A Linguist Explains Vintage Internet Slang" the-toast.net/2015/08/26/wir… /via @pomeranian99

Being Ethical

1. Twitter exists in a strange limbo between private and public communication. While some tweets are Private, the degree to which Public tweets are really “Public" is debatable.
2. Many situations on Twitter are comparable to private conversations in a public space (two friends talking in a restaurant).
3. In situations like this, it’s better to err on the side of caution and work hard to protect and respect individuals’ privacy.
4. General best practices: consult D’Arcy & Young (2012) “*Ethics and social media: Implications for sociolinguistics in the networked public.*"

Twitter API

The way we interact with Twitter data is by requesting data via Twitter's **API** (Application Program Interface).

In order to access the API we'll have to create an application (free and easy) and then authenticate it every time we want to make a query.

API Challenges

While Twitter saves some mind-boggling gigantic numbers of its tweets, through the Public API we can only access tweets going back **a week**.

Twitter also imposes a limit on the amount of calls you can make within a certain time frame (**rate-limiting**). In our examples we won't run into this, but refer to the Advanced Topics Slides for resources for dealing with Rate Limiting.

Constructing a Corpus

If you're considering doing real, publishable work with Twitter, odds are you'll need a corpus. Keep these things in mind when you go about creating a corpus:

1. Dealing with Rate-Limiting
2. Sampling over a long period (usually at least several weeks)
3. Define your scope: location, hashtag, word?
4. Someone might have already constructed a corpus you can use. Network and collaborate!

Setup and Authentication

Since it takes some time, we won't be going over setup and authentication today. Instead we'll be working with data I've already collected.

We'll still be analyzing the code for collecting data but the instructions for setup/authentication are at the end of the slides in the Advanced Topics and Resources section.

Extracting Tweets and Temporal Variation

Initialization

```
#install.packages("packageName")
library(twitteR)
library(ggplot2)
library(maps)
library(stringr)
library(igraph)
library(reshape2)

#setwd("yourDir")
load("twitterWorkshop.rda")
```

```
ls()

## [1] "AFINN"          "sub"            "tweetsDF1"
## [4] "tweetsDF2"      "tweetsDFcities"
```

Center for Feline Friendship

Imagine we're working for the Center for Feline Friendship and want to analyze the most recent English tweets about "cats".

Extraction Basics

If you were gathering data yourself, you'd use the following code:

```
tweets = searchTwitter("cats",lang="en",n=500)
tweetsDF1 = twListToDF(tweets)
```

Extraction Basics

Remember, searches using the public API only go back about a week. If you're using an uncommon search term or parameters, you might get less results than you requested.

```
nrow(tweetsDF1)

## [1] 500

names(tweetsDF1)

##  [1] "text"          "favorited"     "favoriteCount"
##  [4] "replyToSN"     "created"       "truncated"
##  [7] "replyToSID"    "id"            "replyToUID"
## [10] "statusSource"  "screenName"    "retweetCount"
## [13] "isRetweet"     "retweeted"     "longitude"
## [16] "latitude"
```

```
head(tweetsDF1)
```

Activity Spikes

One question you might ask in dealing with temporal variation (whether on the scale of minutes, days, years) is what patterns or structures exist.

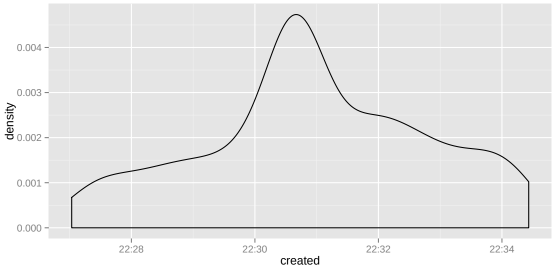
Let's investigate that in our cat data!

Activity Spikes

```
#Remove date information
tweetsDF1$time <- format(tweetsDF1$created, format = "%H:%M:%S")
tweetsDF1$time <- as.POSIXct(tweetsDF1$time, format = "%H:%M:%S")

#Plotting
plot1 <- ggplot(tweetsDF1, aes(created)) + geom_density()
```

```
plot1
```

Activity Spikes

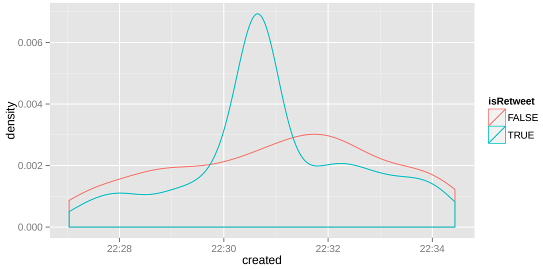


Notice the large spike around 10:31 pm (GMT). What could this be due to? How could we test these two options?

Retweets

It could be an increase in unique tweets containing “cats" or a highly popular retweet (or some combination). Let's investigate

```
plot2 <- ggplot(tweetsDF1, aes(created,color=isRetweet)) + geom_density()
plot2
```



Retweet Frequency

In any network, social interactions are not equally distributed. Some people are more influential than others.

As researchers at the Center for Feline Friendship we're very interested in figuring out what this influential tweet is and who originally made it.

To do that, let's add a new factor to tweetsDF1 which measures the number of times a given tweet or RT occurs in our corpus.

```
tweetsDF1 <- transform(tweetsDF1, freq = ave(seq(nrow(tweetsDF1)),
text,FUN=length))
```

```
head(tweetsDF1[order(-tweetsDF1$freq),])
```

Group Assignment 1

1. What is the most frequent tweet in our corpus? What does its usage of “cats" tell us about the challenges of word senses?
2. Investigate the relationship between **$freq** (occurrence in the corpus) and **$retweetCount**.
   - You'll find it helpful to plot the logged values.
   - How can you interpret **$retweetCount**?
   - What relationship exists between these two values?
   - What does this pattern tell us about our sampling size and networks?

Geotagged Data and Regional Distributions

Geodata Motivation

One interesting question we might ask is how lexical variation is regionally constrained.

Historical Linguistics and Dialectology (among other subdisciplines) are traditionally very interested in isoglosses (be they lexical, phonological, or other).

This is perhaps the most accessible area of linguistics for the layperson. Think of how Josh Katz's maps exploded last year.

For examples of excellent scholarly work on regional lexical variation from written corpora (like Twitter), see the work of Jack Grieve and colleagues.

Let's Go to the Beach

Let's imagine that we want to sample the most recent tweets that contain the word “beach" and also have geo-location data.

Note that one approach would be to take location data from the author's profile and filtering against list of known cities. In this example, however, we'll be working with data that has actual longitude and latitude information from when the tweet was sent.

If you were going to collect the data yourself, you'd do the following. Instead we've already got the data loaded as **tweetsDF2**

```
tweets2 = searchTwitter("beach",lang="en",n=5000)
tweetsDF2 = twListToDF(tweets2)
tweetsDF2 <- tweetsDF2[!is.na(tweetsDF2$longitude),]
```

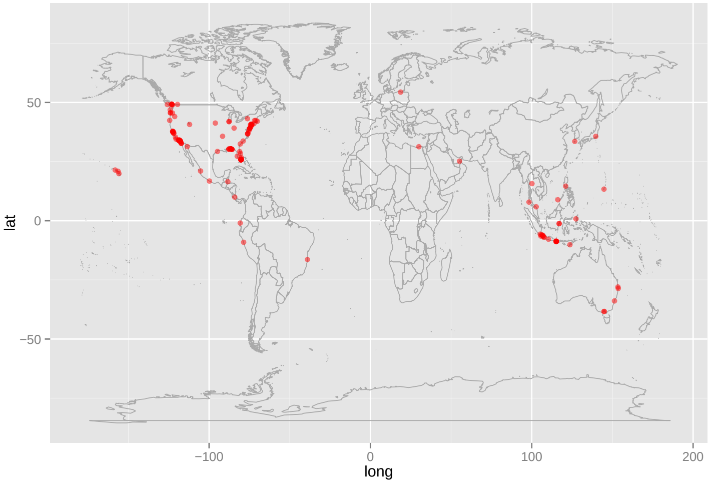
You can also make a search for geo-tagged tweets a certain radius around a given lat/long; see the advanced resources at the end for an example.

Plotting on a Map

```
#Initialize map of Globe
beachMap <- ggplot(map_data("world"))
#Create lines for countries, etc.
beachMap <- beachMap + geom_path(aes(x = long, y = lat, group = group),
color = gray(2/3), lwd = 1/3)
#Add in our geo-tagged data
beachMap <- beachMap + geom_point(data=tweetsDF2,aes(x = longitude,
y = latitude), color = "red", alpha = 1/2, size = 2)
```

```
beachMap
```

Plotting on a Map



Group Assignment 2a

1. Consider the distribution of tweets.
   1.1 Does it match what you'd expect?
   1.2 Why can't we make serious claims about regional usage with the current data?
   1.3 How would you go about testing for an isogloss?
2. The Maps package has a variety of different map options. Let's explore some of them now.
   2.1 Plot our data on a US map instead of a world map. Try changing **map_data("world"))** to **map_data("states"))**
   2.2 What happens when you plot all the data on the US map?
   2.3 To fix this issue, edit the third line of code by changing

```
data=tweetsDF2,
#to
data=tweetsDF2[tweetsDF2$longitude < SOMEVAL &
 tweetsDF2$latitude > SOMEVAL],
```

Group Assignment 2b

3. Now, plot just the data for all the counties in California.
   3.1 See the ‘county‘ example on page 4 of the maps documentation `https://cran.r-project.org/web/packages/maps/maps.pdf`
   3.2 **HINT:** When subsetting by conditions in R you can't do df[5 > x > 1,], you have to do df[5 > x & x > 1,]
4. What areas in California tweet most about the beaches?
5. What are some issues with making this claim from this data?

Sentiment Analysis

What is Sentiment Analysis?

- Can we automatically detect the emotion expressed by an utterance (e.g. positive v. negative) based just on the text?
- Applications for research, industry, marketing, politics, etc...
- Simplest approach (we're taking now): assign values to positive and negative words and add up values.
- More sophisticated approaches: parse utterance into units and learn relationship between sentiment and structure (usually by machine learning techniques: Naive Bayes Classifier, Support Vector Machine, Neural Nets, etc..).

Problems

Computers aren't as good as determining sentiment as people (yet!). Pragmatic, Semantic, and Discursive cues are hard to learn. Consider the difficulty of the following utterance for a value-assigning lexical approach.

"Unlike your company's waffles, your competitor's waffles are delicious, fluffy, and always a good buy!"

A Tale of Two Cities

Imagine that we want to get a feel for how people are tweeting about two cities, Raleigh and Seattle (why did I choose these two and not “Durham" or “Austin"?).

Let’s get the 5000 most recent tweets mentioning each city, add a new factor indicating what city it’s mentioning, and then merge the two data frames into one. If you were going to do it yourself, the code you would use is:

```
tweets3 = searchTwitter('Raleigh',lang="en",n=5000)
tweetsDF3 = twListToDF(tweets3)
tweets4 = searchTwitter('Seattle',lang="en",n=5000)
tweetsDF4 = twListToDF(tweets4)

#Add city factor and merge
tweetsDF3$city <- rep("Raleigh",nrow(tweetsDF3))
tweetsDF4$city <- rep("Seattle",nrow(tweetsDF4))
tweetsDFcities <- rbind(tweetsDF3,tweetsDF4)
```

Methodological Approach

We'll be taking the simplest approach to sentiment analysis: coding words as positive or negative, assigning positive or negative values for words, and then scoring tweets.

To determine negative vs. positive words we'll be using the **AFINN-111** list, used in Lars Kai Hansen, Adam Arvidsson, Finn Årup Nielsen, Elanor Colleoni, Michael Etter, "*Good Friends, Bad News - Affect and Virality in Twitter*", The 2011 International Workshop on Social Computing, Network, and Services (SocialComNet 2011).

Other excellent corpora are LIWC, Wordnet, SentiWordNet, and Senti140 (tweets).

In AFINN, *Breathtaking* = +5; *Bitch* = -5

What's an issue with computing raw scores for all the tweets like this?

Sentiment

“I love it!"

vs.

“I love that the muffins you baked are ready to eat."

Sentiment

“I love it!"

vs.

“I love that the muffins you baked are ready to eat."

We'll want to normalize our raw counts of pos. + neg. words by the total number of words in the utterance/tweet.

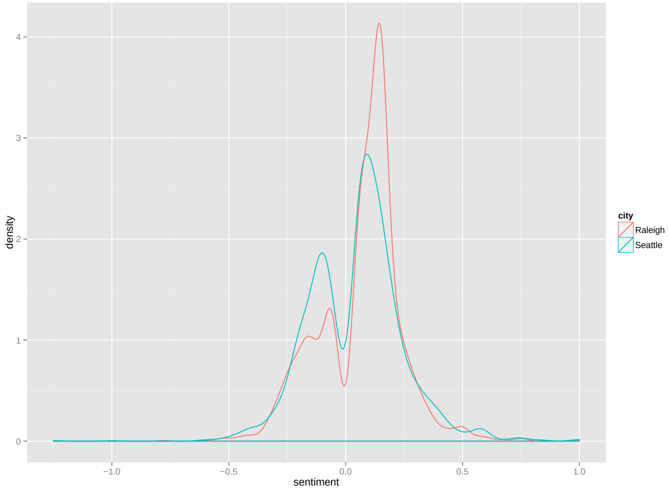
Scoring Sentiment

```r
#Initialize the sentiment vector
tweetsDFcities$sentiment <- rep("NA",nrow(tweetsDFcities))

#Remove problematic UTF-8 symbols and convert text to uppercase
tweetsDFcities$textUP <- str_replace_all(tweetsDFcities$text,"[^[:graph:]]", " ")
tweetsDFcities$textUP <- as.character(toupper(tweetsDFcities$textUP))

#For each tweet get the number of words (defined as number of spaces + 1)
for (i in 1:nrow(tweetsDFcities)){
n_words = str_count(tweetsDFcities$textUP[i]," ") + 1
counter = 0
words <- strsplit(tweetsDFcities$textUP[i]," ")[[1]]

#For each word in the tweet test if it's in our values df and assign the value
for (j in (1:length(words))){
curr_word <- words[j]
if(curr_word %in% AFINN$word){counter = counter + AFINN$value[match(curr_word,AFINN$word)]}
}

#Normalize for length of sentence
tweetsDFcities$sentiment[i] <- counter/n_words
}
tweetsDFcities$sentiment <- as.numeric(tweetsDFcities$sentiment)
```

Plotting Sentiment

```
#The basics
ggplot(tweetsDFcities, aes(sentiment)) + geom_density()

#Let's remove values with sentiment value of 0
ggplot(tweetsDFcities[tweetsDFcities$sentiment != 0,],
       aes(sentiment)) + geom_density()

#What we really want though is the breakdown by city
ggplot(tweetsDFcities[tweetsDFcities$sentiment !=0,],
       aes(sentiment,color=city)) + geom_density()
```

Seattle vs. Raleigh



Group Assignment 3

1. One hypothesis you might have is that retweets will tend to be more positive than regular tweets.
   - Modify the graph code above to test this hypothesis.
   - Do both cities show the same distribution?
   - If not, prove that this is not due to an unequal percentage of retweets per city.
2. Read through some of the most positive and negatively scored tweets. Do their classifications match what you'd expect? What lexical, syntactic, pragmatic, or discursive features lead to misclassifications?

Analyzing Network Data

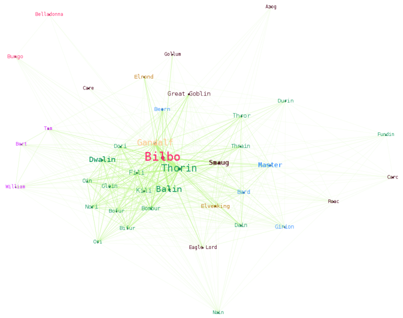
Motivation

Social interactions are not randomly or evenly distributed; people usually interact within systematic structures. In-group cohesion is usually high and ties between groups are less common.

Social Network theory holds that the structure of interactions and relationships governs the flow of information, be it linguistic or social (if that's a distinction we can make).

Implications for social theory, language change and contact, sociolinguistics, and more.

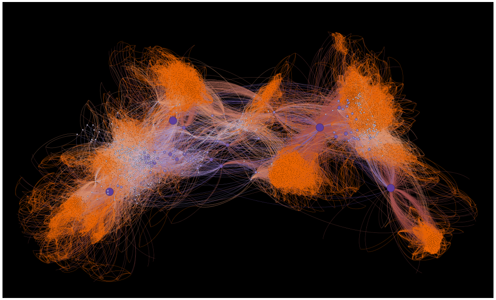
The *Hobbit* Network Graph



https://www.reddit.com/r/dataisbeautiful/comments/2x5fte/social_network_of_tolkiens_hobbit_characters_oc/

A Tie indicates two characters occurred on the same page. Size of label correlates to number of total occurrences.

College Friends



http://imgur.com/HEvgdm9

4 roommates from 2 separate colleges; smaller nodes are sized by how many of the 4 roommates they are connected to.

Social Networks and Linguistics

Network analysis got its first introduction to sociolinguistics in the 80s with work by Milroy & Milroy.

Since then we've seen an appreciable body of literature dealing with ethnographic descriptions of smaller networks (e.g. Penny Eckert and Burnouts v. Jocks) or broader analyses of larger networks (e.g. Robin Dodsworth's work in Raleigh).

Some Terms

1. **Vertex** - actors in your analysis (usually individuals, can be organizations, groups, etc.); may be divided into primary and secondary nodes
2. **Edge** - ties connecting nodes in your network; may be defined in a variety of ways
3. **Network Measures**
    3.1 **Centrality/Betweenness/Closeness** - roughly, how important is an individual node to the network? Do they provide links between nodes that wouldn't exist if they weren't there? Do they connect many other nodes? Do they significantly reduce the number of connections needed to get from one node to another?
    3.2 **Cohesion** - How tightly connected is the network? Specifically, how many ties would have to be removed to separate the group into two groups?
    3.3 **Multiplexity/Density** - Related to cohesion, but more familiar to linguists. Roughly, how connected are individuals in the network? Dense, multiplex networks are those with many shared ties between nodes.

Following Network

- We're interested in analyzing networks between twitter users, the people they follow, and the people these people follow.
- For this, we'll be extracting all the people I follow on twitter, then extracting all the people they follow.
- In order to be ethical about usage, all twitter handles are anonymized.
- If you were going to carry out the extraction yourself, you'd use the code on the next slide.
- This is a very long operation and you'll likely want to set up the script to run remotely. See the advanced resources slides for an example.

```r
curr <- getUser("eric_wilbanks")
#Get list of who I follow
following <- do.call('rbind', lapply(curr$getFriends(),as.data.frame))

#Inititalize full DF
full <- NULL

#Loop over users in the Following df
for (i in 1:nrow(following)){
sub <- NULL

#Wait 60 seconds to prevent rate limiting
Sys.sleep(60)

#Get who the current user is following (as in the above example)
curr <- getUser(following[i,11])
sub <- do.call('rbind', lapply(curr$getFriends(),as.data.frame))

#Record who the current user is in the original factor
sub$original <- rep(curr$screenName,nrow(sub))

#Bind the subset with the full df
full <- rbind(full,sub)}
```

Generating Network Graphs

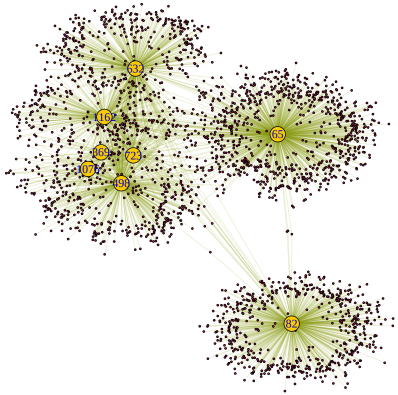
- To generate our network graph, we'll use the *igraph* package.
- The algorithms for creating graphs are very resource-intense, for the current workshop we'll be looking at a subset of the original data frame called **sub**.
- iGraph expects a matrix of relationships like the following example with 1's indicating ties between nodes.

	A	**B**	**C**
A	0	1	1
B	1	0	0
C	1	0	0

```r
#Convert the data.frame into a matrix for doing network calculations
sub2 <- dcast(sub, screenName ~ original, fun.aggregate=length)
sub.matrix <- as.matrix(sub2[,2:length(sub2)])

#Change the rownames to be screenNames
row.names(sub.matrix) <- sub2$screenName

#Create the igraph object
sub.igraph <- graph.incidence(sub.matrix, mode="in")

#Set graphics options for Vertices and Edges
len <- length(V(sub.igraph))
break1 <- len - length(sub2) + 1
break2 <- break1 + 1
E(sub.igraph)$color = rgb(138,155,15,50,maxColorValue=255)
V(sub.igraph)$color[1:break1] = rgb(189,21,80,maxColorValue=255)
V(sub.igraph)$color[break2:len] = rgb(248,202,0,maxColorValue=255)
V(sub.igraph)$size[1:break1] = 1
V(sub.igraph)$size[break2:len] = 8
V(sub.igraph)$label <- V(sub.igraph)$name
V(sub.igraph)$label[1:break1] <- NA
V(sub.igraph)$label.color[break2:len] = rgb(0,0,204,maxColorValue=255)
```

```r
#Plot the network structure and save to a PDF
pdf(file = "output1.pdf")
plot(sub.igraph, layout=layout.fruchterman.reingold,vertex.label.color = V(sub.igraph)$label.color)
dev.off()
```

Network Graph



Group Assignment 4

1. Recall that our information about followers is directional (A follows B, not necessarily vice versa).
   - Consult the documentation for graph.incidence `http://igraph.org/r/doc/graph_from_incidence_matrix.html` and edit the graph so it has arrows.
   - Try changing the size and width of the arrows. `http://igraph.org/r/doc/plot.common.html`
2. Besides visualizing the data, we can also query the network structure directly for measures of centrality, etc.
   - Consult the manual for the Eigenvector Centrality measure (`http://igraph.org/r/doc/eigen_centrality.html`) and then calculate the centrality over sub.igraph (you'll want to store the output as an object)
   - Now, change the size of the vertices in the graph to correspond to their centrality. (**HINT:** consider multiplying the centrality by some constant)
   - What observations can you make looking at the new network? Looking at centrality, which of the 8 main nodes were used to seed the network (i.e. which node is my twitter handle)?

That's All Folks!

Thank you!

Questions? Comments? Concerns?

Advanced Topics and Resources

Setting Up an App

To interface with the Twitter Public API, you'll need to create an account and then use that account to create an App.

1. Go to `twitter.com` and create an account or use your own (it can be private/unsearchable).
2. Go to `apps.twitter.com` and select 'Create New App'.
   - Give your application some unique name ('Linguistics Data Testing'), a short description, and then a placeholder url for the 'Website' field.
   - Read through the developer user agreement and then 'Create your Twitter Application'.
3. Now, click 'Keys and Access Tokens' and then 'Create Access Token'.
   - You should see several fields of randomly generated strings for 'Consumer Key', 'Consumer Secret', 'Access Token', and 'Access Token Secret'
   - Do **NOT** share these keys or otherwise make them publicy known, otherwise malicious individuals could access your account.

Establishing Credentials in an .Renviron file

We'll need to authenticate each new R session but don't want to worry about having to remove our passwords and access keys from our scripts. To avoid this, we'll store this information in an **.Renviron** file external to the script and R session and then call on these variables.

Use the following code to create your **.Renviron** file; you only need to do this once! You'll have to reload your R session after you create/make changes to your .Renviron file.

```
credentials <- c(
"twitter_api_key = API_KEY_HERE",
"twitter_api_secret = API_SECRET_HERE",
"twitter_access_token =  ACCESS_TOKEN_HERE",
"twitter_access_secret = ACCESS_SECRET_HERE"
"pw = PASSWORD_HERE")

fname <- paste0(normalizePath("~/"),".Renviron")
writeLines(credentials, fname)
```

Authenticating a Session

For every new R session that you want to make calls through the twitteR package, you'll have to draw on the information stored in the **.Renviron** file by using the following code.

```
pw <- Sys.getenv("pw")
api_key <- Sys.getenv("twitter_api_key")
api_secret <- Sys.getenv("twitter_api_secret")
access_token <- Sys.getenv("twitter_access_token")
access_secret <- Sys.getenv("twitter_access_secret")
setup_twitter_oauth(api_key,api_secret,access_token,access_secret)
1
```

Dealing with Rate Limiting

Since we have limits on the number of calls to the Twitter API we can make in a certain time span, you'll have to spread out your requests. If you're looking to create a large corpus, this could involve requesting data over several days or weeks.

For long-running scripts like this you'll likely want to host the process remotely on a server or networked computer and then have the script notify you when it's complete.

On the following page I've shared a version of a script I've used for collecting large quantities of network data over several days.

A more sophisticated and robust approach would involve making calls to **getCurRateLimitInfo()** from the **twitteR** package and using this information to determine when to pause, rather than a flat **Sys.sleep()** every iteration.

When operating on such a time scale, you'll also need to build in error-catching clauses and the ability to safely append data every iteration, rather than writing at the end. What happens if after you collect your initial tweets/users to iterate over, one of the tweets/users in your data frame is deleted? Your script fails at hour 15 and you have to start all over, that's what happens. I leave the prudent error programming as an exercise for you.

```r
library(twitteR)
library(mailR)
#Search For Specific Term
term = "MySearchTerm"
tweets = searchTwitter(term, lang="en",n=1000)
tweets_df = twListToDF(tweets)
file_name = paste(term,"_tweets.csv",sep="")
#Initialize DF
full <- NULL
#Loop over all tweets in DF
for (i in 1:nrow(tweets_df)){
sub <- NULL
#Pause to prevent Rate Limiting
Sys.sleep(90)
#For each tweet, get the user who generated it
curr <- getUser(tweets_df[i,11])
#Get the users that that user is following
sub <- do.call('rbind', lapply(curr$getFriends(),as.data.frame))
sub$original <- rep(curr$screenName,nrow(sub))
#Append that list of followers to the main DF
full <- rbind(full,sub)
#If we're done, save the data frame and send an email to ourselves
if (i == nrow(tweets_df)){
write.csv(full,file_name,row.names=F);
send.mail(from="emailAddress",to="emailAddress",subject="Your Script is Done Processing!",body="It's done!",
smtp=list(host.name="myEmailSMTP", port=465, user.name="emailAddress", passwd="emailPW",ssl=TRUE),
authenticate=TRUE, send=TRUE)
}}
```

Search by Location

Besides searching for a query and then filtering to get at geo-coded tweets like we saw in this workshop, you could also search for geo-coded tweets within a certain radius of a given point.

Here's an example for the 1000 most recent geo-coded tweets within 5mi of the center of Raleigh. Notice that since we can't query without a search term, I'm searching for tweets containing a space.

The **geocode** argument has the syntax **geocode="lat,long,radius"**.

```
searchTwitter(" ", lang="en", geocode="35.7806,-78.6389,5mi", n=1000)
```

Custom Maps

You can also generate custom maps from your own images our figures, provided you have the relevant latitude and longitude information. Raleigh tweets anyone?

Ryan Garner provides a great example at the following link:

`http://blog.revolutionanalytics.com/2015/01/creating-a-custom-soil-attribute-plot-using-ggmap.html`

anonymizeColumns.R

Here's a function that I wrote to anonymize the data in the network example. To use it, first source() or copy/paste the function into your R session. Then, call it by doing: **newDF >- anonymizeColumns(oldDF, c(column1,column2))**

```
anonymizeColumns <- function(df, colIDs) {

        id <- if(is.character(colIDs)) match(colIDs, names(df)) else colIDs
        anonContainer <- NULL

        for(id in colIDs) {
                anonContainer <- c(anonContainer,as.character(df[[id]]))
        }

        anonContainer <- unique(anonContainer)
        keyDF <- data.frame(anonContainer, as.character(as.numeric(as.factor(anonContainer))))
        names(keyDF) <- c('key','keyVal')
        keyDF$key <- as.character(keyDF$key)
        keyDF$keyVal <- as.character(keyDF$keyVal)

        for(id in colIDs) {
                for(id2 in 1:nrow(keyDF)){
                df[[id]] <- as.character(df[[id]])
                df[[id]][df[[id]] %in% keyDF$key[id2]] <- keyDF$keyVal[id2]
                }
                df[[id]] <- as.factor(df[[id]])
        }
        df
}
```