Real-time network analysis in R using Twitter

Drew Conway

New York University — Department of Politics

July 22, 2010

Standing on the shoulders of giants

Collection and parsing of Twitter data

- `twitteR`
- Jeff Gentry, Dept. of, Biostatistics — Harvard University

Standing on the shoulders of giants

Collection and parsing of Twitter data

- `twitteR`
- Jeff Gentry, Dept. of, Biostatistics — Harvard University

Graph representations, analysis and visualization

- `igraph`
- Gábor Csárdi, Dept. of Biophysics — Université de Lausanne (Switzerland)

Standing on the shoulders of giants

Collection and parsing of Twitter data

- `twitteR`
- Jeff Gentry, Dept. of, Biostatistics — Harvard University

Graph representations, analysis and visualization

- `igraph`
- Gábor Csárdi, Dept. of Biophysics — Université de Lausanne (Switzerland)

Data visualization

- ggplot2
- Hadley Wickham, Dept. of Statistics — Rice University

A bit on tools

The number of software suites and packages available for conducting social network analysis has exploded over the past ten years

- In general, this software can be categorized in two ways:

A bit on tools

The number of software suites and packages available for conducting social network analysis has exploded over the past ten years

- In general, this software can be categorized in two ways:
  - **Type** - many SNA tools are developed to be standalone applications, while others are language specific packages
  - **Intent** - consumers and producer of SNA come from a wide range of technical expertise and/or need, therefore, there exist simple tools for data collection and basic analysis, as well as complex suites for advanced research

A bit on tools

The number of software suites and packages available for conducting social network analysis has exploded over the past ten years

- In general, this software can be categorized in two ways:
  - **Type** - many SNA tools are developed to be standalone applications, while others are language specific packages
  - **Intent** - consumers and producer of SNA come from a wide range of technical expertise and/or need, therefore, there exist simple tools for data collection and basic analysis, as well as complex suites for advanced research

	Standalone Apps	*Modules & Packages*
Basic	- ORA (Windows) - Analyst Notebook (Windows) - KrakPlot (Windows)	- libSNA (Python) - UrlNet (Python) - NodeXL (MS Excel)
Advanced	- UCINet (Windows) - Pajek (Multi) - Network Workbench (Multi)	- NetworkX (Python) - JUNG (Java) - igraph (Python, R & Ruby)

A bit on tools

The number of software suites and packages available for conducting social network analysis has exploded over the past ten years

- In general, this software can be categorized in two ways:
  - **Type** - many SNA tools are developed to be standalone applications, while others are language specific packages
  - **Intent** - consumers and producer of SNA come from a wide range of technical expertise and/or need, therefore, there exist simple tools for data collection and basic analysis, as well as complex suites for advanced research

	Standalone Apps	*Modules & Packages*
Basic	- ORA (Windows) - Analyst Notebook (Windows) - KrakPlot (Windows)	- libSNA (Python) - UrlNet (Python) - NodeXL (MS Excel)
Advanced	- UCINet (Windows) - Pajek (Multi) - Network Workbench (Multi)	- NetworkX (Python) - JUNG (Java) - igraph (Python, R & Ruby)

Many of the above tools have visualization components, but several tools are designed specifically for visualization: Graphviz, NetDraw, Tom Sawyer, Gephi, etc.

A bit on tools

The number of software suites and packages available for conducting social network analysis has exploded over the past ten years

- In general, this software can be categorized in two ways:
  - **Type** - many SNA tools are developed to be standalone applications, while others are language specific packages
  - **Intent** - consumers and producer of SNA come from a wide range of technical expertise and/or need, therefore, there exist simple tools for data collection and basic analysis, as well as complex suites for advanced research

	Standalone Apps	*Modules & Packages*
Basic	- ORA (Windows) - Analyst Notebook (Windows) - KrakPlot (Windows)	- libSNA (Python) - UrlNet (Python) - NodeXL (MS Excel)
Advanced	- UCINet (Windows) - Pajek (Multi) - Network Workbench (Multi)	- NetworkX (Python) - JUNG (Java) - igraph (Python, R & Ruby)

Many of the above tools have visualization components, but several tools are designed specifically for visualization: Graphviz, NetDraw, Tom Sawyer, Gephi, etc. What I use

Pros and Cons of SNA in R

Pros

Diversity of tools available in R

- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Cons

Pros and Cons of SNA in R

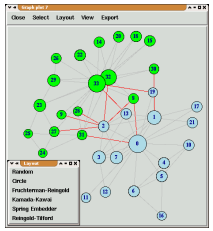
Pros

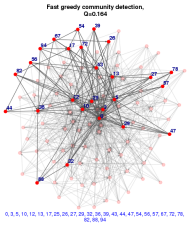
Diversity of tools available in R

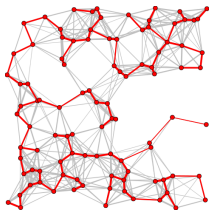
- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Built-in visualization tools

- Take advantage of R's built-in graphics tools







Cons

Pros and Cons of SNA in R

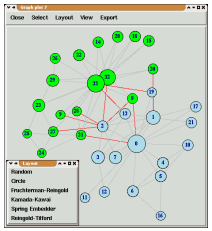
Pros

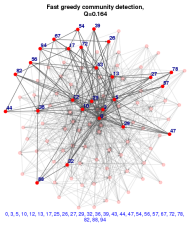
Diversity of tools available in R

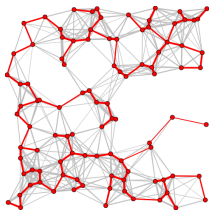
- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Built-in visualization tools

- Take advantage of R's built-in graphics tools







Immediate access to more stats

- Perform SNA and network based econometrics "under the same roof"

Cons

Pros and Cons of SNA in R

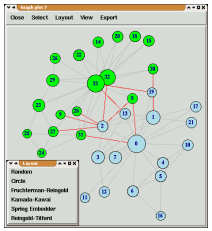
Pros

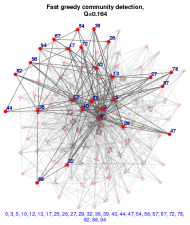
Diversity of tools available in R

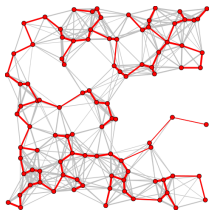
- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Built-in visualization tools

- Take advantage of R's built-in graphics tools







Immediate access to more stats

- Perform SNA and network based econometrics "under the same roof"

Cons

Steep learning curve for SNA novices

- As with most things in R, the network analysis packages were designed by analysts for analysts
- These tools require at least a moderate familiarity with network structures and basic metrics

Pros and Cons of SNA in R

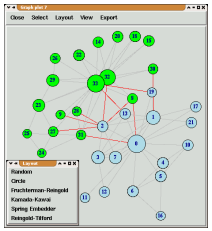
Pros

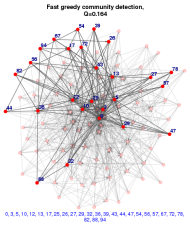
Diversity of tools available in R

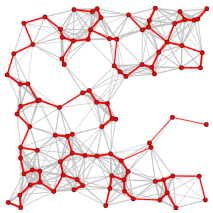
- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Built-in visualization tools

- Take advantage of R's built-in graphics tools







Immediate access to more stats

- Perform SNA and network based econometrics "under the same roof"

Cons

Steep learning curve for SNA novices

- As with most things in R, the network analysis packages were designed by analysts for analysts
- These tools require at least a moderate familiarity with network structures and basic metrics

Structural Holes

Burt's constraint is higher if ego has less, or mutually stronger related (i.e. more redundant) contacts. Burt's measure of constraint, C[i], of vertex i's ego network V[i]

Pros and Cons of SNA in R

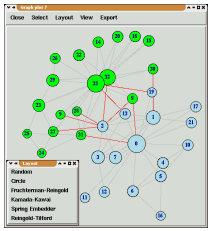
Pros

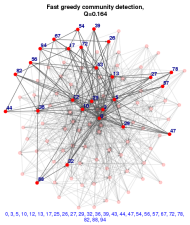
Diversity of tools available in R

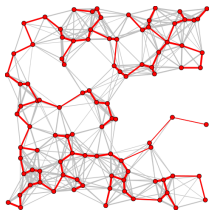
- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Built-in visualization tools

- Take advantage of R's built-in graphics tools







Immediate access to more stats

- Perform SNA and network based econometrics "under the same roof"

Cons

Steep learning curve for SNA novices

- As with most things in R, the network analysis packages were designed by analysts for analysts
- These tools require at least a moderate familiarity with network structures and basic metrics

Structural Holes

Burt's constraint is higher if ego has less, or mutually stronger related (i.e. more redundant) contacts. Burt's measure of constraint, C[i], of vertex i's ego network V[i]

In ability to exchange data between other network analysis packages

- `statnet` library powerful for modeling
- `igraph` and `statnet` represent networks in different ways
- Difficult to exchange data between libraries

Comparing two network metrics to find key actors

Often social network analysis is used to identify key actors within a social group. To identify these actors, various centrality metrics can be computed based on a network's structure

- Degree (number of connections)
- Betweenness (number of shortest paths an actor is on)
- Closeness (relative distance to all other actors)
- Eigenvector centrality (leading eigenvector of sociomatrix)

Comparing two network metrics to find key actors

Often social network analysis is used to identify key actors within a social group. To identify these actors, various centrality metrics can be computed based on a network's structure

- Degree (number of connections)
- Betweenness (number of shortest paths an actor is on)
- Closeness (relative distance to all other actors)
- Eigenvector centrality (leading eigenvector of sociomatrix)

One method for using these metrics to identify key actors is to plot actors' scores for Eigenvector centrality versus Betweenness. Theoretically, these metrics should be approximately linear; therefore, any non-linear outliers will be of note.

- An actor with very high betweenness but low EC may be a critical gatekeeper to a central actor
- Likewise, an actor with low betweenness but high EC may have unique access to central actors

Finding Key Actors with R

For this example, we will use the main component of the social network collected on drug users in Hartford, CT.[1] The network has 194 nodes and 273 edges.

[1] Weeks, et al (2002) http://dx.doi.org/10.1023/A:1015457400897

Finding Key Actors with R

For this example, we will use the main component of the social network collected on drug users in Hartford, CT.[1] The network has 194 nodes and 273 edges.

Load the data into igraph

```
library(igraph)
G<-read.graph("drug_main.txt",format="edgelist")
G<-as.undirected(G)
# By default, igraph inputs edgelist data as a directed graph.
# In this step, we undo this and assume that all relationships are reciprocal.
```

[1] Weeks, et al (2002) http://dx.doi.org/10.1023/A:1015457400897

Finding Key Actors with R

For this example, we will use the main component of the social network collected on drug users in Hartford, CT.[1] The network has 194 nodes and 273 edges.

Load the data into igraph

```
library(igraph)
G<-read.graph("drug_main.txt",format="edgelist")
G<-as.undirected(G)
# By default, igraph inputs edgelist data as a directed graph.
# In this step, we undo this and assume that all relationships are reciprocal.
```

Store metrics in new data frame

```
cent<-data.frame(bet=betweenness(G),eig=evcent(G)$vector)
# evcent returns lots of data associated with the EC, but we only need the
# leading eigenvector
res<-lm(eig~bet,data=cent)$residuals
cent<-transform(cent,res=res)
# We will use the residuals in the next step
```

[1] Weeks, et al (2002) http://dx.doi.org/10.1023/A:1015457400897

Finding Key Actors with R

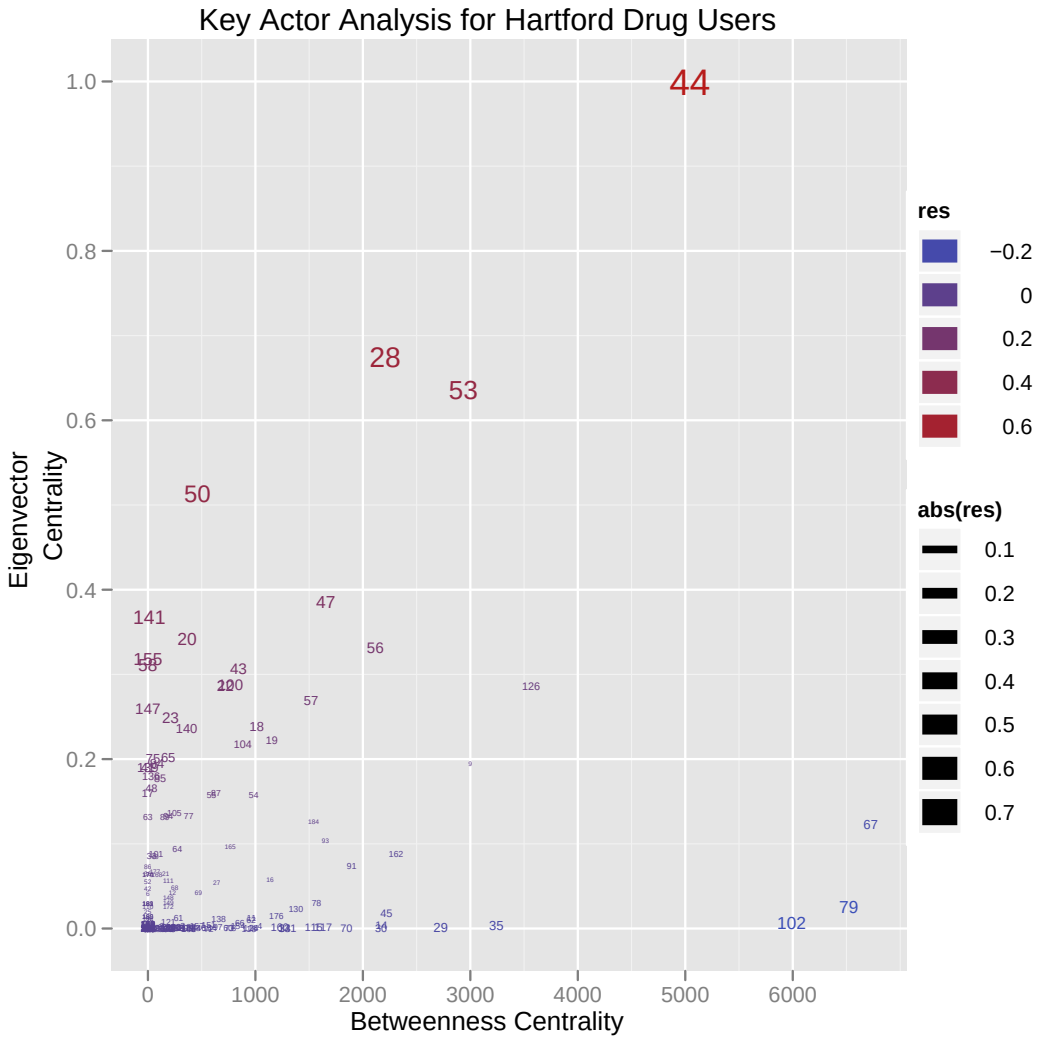
Plot the data

```
library(ggplot2)
# We use ggplot2 to make things a
# bit prettier
p<-ggplot(cent,aes(x=bet,y=eig,
    label=rownames(cent),colour=res,
    size=abs(res)))+xlab("Betweenness
    Centrality")+ylab("Eigenvector
    Centrality")
# We use the residuals to color and
# shape the points of our plot,
# making it easier to spot outliers.
p+geom_text()+opts(title="Key Actor
    Analysis for Hartford Drug Users")
# We use the geom_text function to plot
# the actors' ID's rather than points
# so we know who is who
```

Finding Key Actors with R

Plot the data

```
library(ggplot2)
# We use ggplot2 to make things a
# bit prettier
p<-ggplot(cent,aes(x=bet,y=eig,
    label=rownames(cent),colour=res,
    size=abs(res)))+xlab("Betweenness
    Centrality")+ylab("Eigenvector
    Centrality")
# We use the residuals to color and
# shape the points of our plot,
# making it easier to spot outliers.
p+geom_text()+opts(title="Key Actor
    Analysis for Hartford Drug Users")
# We use the geom_text function to plot
# the actors' ID's rather than points
# so we know who is who
```



Where to get social graph data

Recently, there has been an explosion of resources for scraping social graph

Service	Data	API Docs
twitter	Following(ers), @-replies, date/time/geo	http://apiwiki.twitter.com/
facebook	Friends, Wall Posts, date/time	http://developers.facebook.com/docs/api
Google	All SocialGraph relationships	http://code.google.com/apis/socialgraph/
foursquare	Friends, Check-ins	http://foursquare.com/developers/
hunch	"Taste graph", recommendations	http://hunch.com/developers/
The New York Times	Congressional votes, campaign finance	http://developer.nytimes.com/docs

Where to get social graph data

Recently, there has been an explosion of resources for scraping social graph

Service	Data	API Docs
twitter	Following(ers), @-replies, date/time/geo	http://apiwiki.twitter.com/
facebook	Friends, Wall Posts, date/time	http://developers.facebook.com/docs/api
Google	All SocialGraph relationships	http://code.google.com/apis/socialgraph/
foursquare	Friends, Check-ins	http://foursquare.com/developers/
hunch	"Taste graph", recommendations	http://hunch.com/developers/
The New York Times	Congressional votes, campaign finance	http://developer.nytimes.com/docs

There is clearly no shortage of data

- Each service provides different relational context
- Data formats are generally JSON, Atom, XML, or some combination
- For a more extensive list of API resources, see HackNY wiki of local startups

Why use Twitter for analyzing networks?

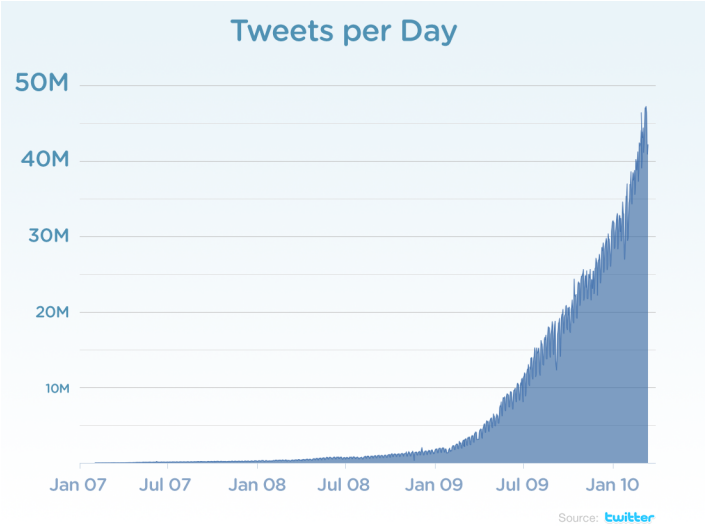
Unprecedented scale and accessibility

- Twitter provides free and open (rate-limited) access to their data
- The amount of data pushed to daily is enormous
- "Following" structure natural social network

Why use Twitter for analyzing networks?

Unprecedented scale and accessibility

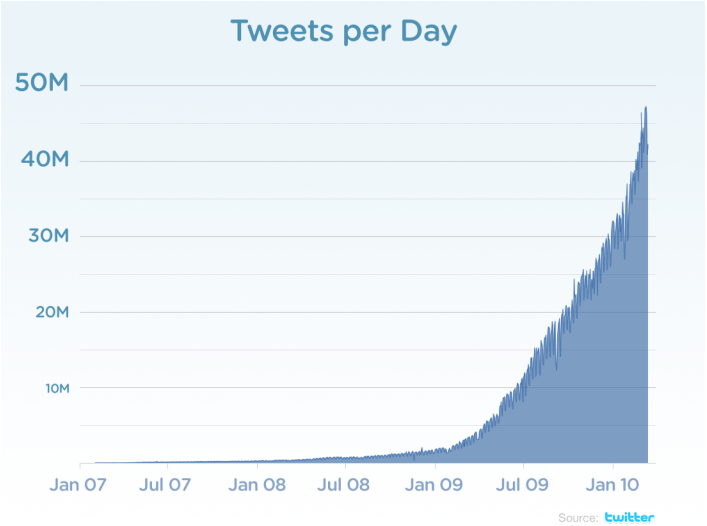
- Twitter provides free and open (rate-limited) access to their data
- The amount of data pushed to daily is enormous
- "Following" structure natural social network



Why use Twitter for analyzing networks?

Unprecedented scale and accessibility

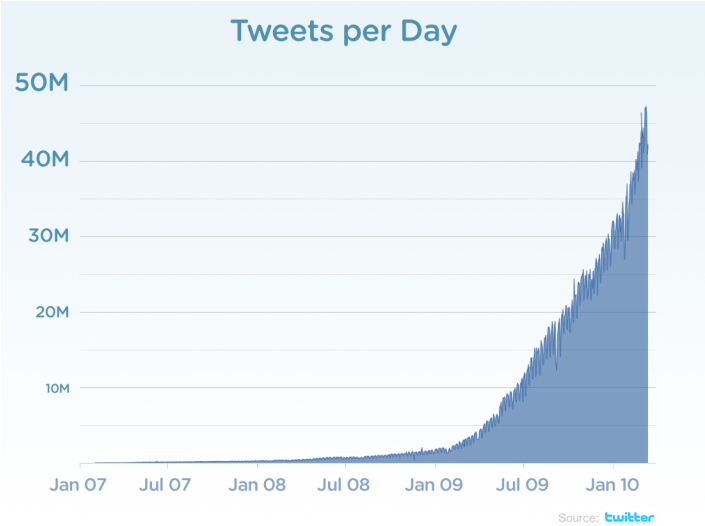
- Twitter provides free and open (rate-limited) access to their data
- The amount of data pushed to daily is enormous
- "Following" structure natural social network



Rich meta-data

- Context of relationships can be inferred by Tweet content
- All relationships longitudinal
- Many Tweets contain geospatial data

Why use Twitter for analyzing networks?



Unprecedented scale and accessibility

- Twitter provides free and open (rate-limited) access to their data
- The amount of data pushed to daily is enormous
- "Following" structure natural social network

Rich meta-data

- Context of relationships can be inferred by Tweet content
- All relationships longitudinal
- Many Tweets contain geospatial data

Easy integration into R

- Data easily parsed with `twitteR` library (@geoffjentry)
- Quickly build edgelist structures
- Can use libraries such as `tm` and `ts` for text and longitudinal analysis

Infochimps.org

In addition to pulling the data yourself, Infochimps.org also offers large data sets for download (for a fee)

- Monthly Twitter census data

Infochimps also has a set of internally defined Twitter metrics that can be accessed freely via an API (`http://api.infochimps.com/`)

- `trstrank` — a version of Google's PageRank for Twitter
- Influence metrics — Twitter user "retweets" and @ replies

Getting started

Load libraries and initialize Twitter session

```
# We will use ggplot2 for later visualization
library(twitteR)
library(igraph)
library(ggplot2)

# Initialize a session to gather dat
user<-"your.name"
pass<-"your.pass"
twit<-initSession(user,pass)

# Get all the users from the last 100 tweets containing some hashtag
hashtag<-"my.hashtag"
hash.tweets<-get.hashtag(hashtag)
hash.users<-users.from.statuses(hash.tweets)
```

Getting started

Load libraries and initialize Twitter session

```
# We will use ggplot2 for later visualization
library(twitteR)
library(igraph)
library(ggplot2)

# Initialize a session to gather dat
user<-"your.name"
pass<-"your.pass"
twit<-initSession(user,pass)

# Get all the users from the last 100 tweets containing some hashtag
hashtag<-"my.hashtag"
hash.tweets<-get.hashtag(hashtag)
hash.users<-users.from.statuses(hash.tweets)
```

Getting started

Load libraries and initialize Twitter session

```
# We will use ggplot2 for later visualization
library(twitteR)
library(igraph)
library(ggplot2)

# Initialize a session to gather dat
user<-"your.name"
pass<-"your.pass"
twit<-initSession(user,pass)

# Get all the users from the last 100 tweets containing some hashtag
hashtag<-"my.hashtag"
hash.tweets<-get.hashtag(hashtag)
hash.users<-users.from.statuses(hash.tweets)
```

Getting started

Load libraries and initialize Twitter session

```
# We will use ggplot2 for later visualization
library(twitteR)
library(igraph)
library(ggplot2)

# Initialize a session to gather dat
user<-"your.name"
pass<-"your.pass"
twit<-initSession(user,pass)

# Get all the users from the last 100 tweets containing some hashtag
hashtag<-"my.hashtag"
hash.tweets<-get.hashtag(hashtag)
hash.users<-users.from.statuses(hash.tweets)
```

Create several helper functions to generate network data from Twitter

```
# Function for converting twitteR object data to vector
friend.vector<-function(friends) {
    v<-c()
    for(i in 1:length(friends)) {v<-append(v,as.character(screenName(friends[[i]])))}
    return(v)}
# Function for generating edgelists
create.adj<-function(seed) {
    u<-getUser(seed)
    # Cannot create network data from protected accts
    if(protected(u)==FALSE) {
        outdegree<-userFriends(u,twit)
        friends<-friend.vector(outdegree)
        return(list(adj.list=cbind(seed,friends),seed.friends=friends))
    }
    else {
        return(list(adj.list=NA,seed.friends=NA))
    }}
# Get the last 100 tweets containing the given hashtag
get.hashtag<-function(hashtag,session=getCurlHandle()) {
    search.url<-paste("http://search.twitter.com/search.json?q=%23",hashtag,"&rpp=100",sep="")
    out <- getURL(search.url, curl = session)
    jsonList <- twFromJSON(out)[[1]]
    return(sapply(jsonList, buildStatus))}
# Get the users from a list of tweets
users.from.statuses<-function(status.list) {
    users<-c()
    for(i in 1:length(status.list)) {
        users<-append(users,(screenName(status.list[[i]])))
    }
    return(unique(users))}
```

Create several helper functions to generate network data from Twitter

```
# Function for converting twitteR object data to vector
friend.vector<-function(friends) {
    v<-c()
    for(i in 1:length(friends)) {v<-append(v,as.character(screenName(friends[[i]])))}
    return(v)}
# Function for generating edgelists
create.adj<-function(seed) {
    u<-getUser(seed)
    # Cannot create network data from protected accts
    if(protected(u)==FALSE) {
        outdegree<-userFriends(u,twit)
        friends<-friend.vector(outdegree)
        return(list(adj.list=cbind(seed,friends),seed.friends=friends))
    }
    else {
        return(list(adj.list=NA,seed.friends=NA))
    }}
# Get the last 100 tweets containing the given hashtag
get.hashtag<-function(hashtag,session=getCurlHandle()) {
    search.url<-paste("http://search.twitter.com/search.json?q=%23",hashtag,"&rpp=100",sep="")
    out <- getURL(search.url, curl = session)
    jsonList <- twFromJSON(out)[[1]]
    return(sapply(jsonList, buildStatus))}
# Get the users from a list of tweets
users.from.statuses<-function(status.list) {
    users<-c()
    for(i in 1:length(status.list)) {
        users<-append(users,(screenName(status.list[[i]])))
    }
    return(unique(users))}
```

Create several helper functions to generate network data from Twitter

```
# Function for converting twitteR object data to vector
friend.vector<-function(friends) {
    v<-c()
    for(i in 1:length(friends)) {v<-append(v,as.character(screenName(friends[[i]])))}
    return(v)}
# Function for generating edgelists
create.adj<-function(seed) {
    u<-getUser(seed)
    # Cannot create network data from protected accts
    if(protected(u)==FALSE) {
        outdegree<-userFriends(u,twit)
        friends<-friend.vector(outdegree)
        return(list(adj.list=cbind(seed,friends),seed.friends=friends))
    }
    else {
        return(list(adj.list=NA,seed.friends=NA))
    }}
# Get the last 100 tweets containing the given hashtag
get.hashtag<-function(hashtag,session=getCurlHandle()) {
    search.url<-paste("http://search.twitter.com/search.json?q=%23",hashtag,"&rpp=100",sep="")
    out <- getURL(search.url, curl = session)
    jsonList <- twFromJSON(out)[[1]]
    return(sapply(jsonList, buildStatus))}
# Get the users from a list of tweets
users.from.statuses<-function(status.list) {
    users<-c()
    for(i in 1:length(status.list)) {
        users<-append(users,(screenName(status.list[[i]])))
    }
    return(unique(users))}
```

Create several helper functions to generate network data from Twitter

```
# Function for converting twitteR object data to vector
friend.vector<-function(friends) {
    v<-c()
    for(i in 1:length(friends)) {v<-append(v,as.character(screenName(friends[[i]])))}
    return(v)}
# Function for generating edgelists
create.adj<-function(seed) {
    u<-getUser(seed)
    # Cannot create network data from protected accts
    if(protected(u)==FALSE) {
        outdegree<-userFriends(u,twit)
        friends<-friend.vector(outdegree)
        return(list(adj.list=cbind(seed,friends),seed.friends=friends))
    }
    else {
        return(list(adj.list=NA,seed.friends=NA))
    }}
# Get the last 100 tweets containing the given hashtag
get.hashtag<-function(hashtag,session=getCurlHandle()) {
    search.url<-paste("http://search.twitter.com/search.json?q=%23",hashtag,"&rpp=100",sep="")
    out <- getURL(search.url, curl = session)
    jsonList <- twFromJSON(out)[[1]]
    return(sapply(jsonList, buildStatus))}
# Get the users from a list of tweets
users.from.statuses<-function(status.list) {
    users<-c()
    for(i in 1:length(status.list)) {
        users<-append(users,(screenName(status.list[[i]])))
    }
    return(unique(users))}
```

Create several helper functions to generate network data from Twitter

```
# Function for converting twitteR object data to vector
friend.vector<-function(friends) {
    v<-c()
    for(i in 1:length(friends)) {v<-append(v,as.character(screenName(friends[[i]])))}
    return(v)}
# Function for generating edgelists
create.adj<-function(seed) {
    u<-getUser(seed)
    # Cannot create network data from protected accts
    if(protected(u)==FALSE) {
        outdegree<-userFriends(u,twit)
        friends<-friend.vector(outdegree)
        return(list(adj.list=cbind(seed,friends),seed.friends=friends))
    }
    else {
        return(list(adj.list=NA,seed.friends=NA))
    }}
# Get the last 100 tweets containing the given hashtag
get.hashtag<-function(hashtag,session=getCurlHandle()) {
    search.url<-paste("http://search.twitter.com/search.json?q=%23",hashtag,"&rpp=100",sep="")
    out <- getURL(search.url, curl = session)
    jsonList <- twFromJSON(out)[[1]]
    return(sapply(jsonList, buildStatus))}
# Get the users from a list of tweets
users.from.statuses<-function(status.list) {
    users<-c()
    for(i in 1:length(status.list)) {
        users<-append(users,(screenName(status.list[[i]])))
    }
    return(unique(users))}
```

Build the network

```
# We now build the network, but we have to make sure the initial seed is not protected
seed.user<-1
while(protected(getUser(hash.users[seed.user]))) {seed.user<-seed.user+1}
# Now create the edgelists for all uses
for(u in seed.user:length(hash.users)) {
    if(u==seed.user) {
        hash.list<-create.adj(hash.users[u])
        hash.el<-hash.list$adj.list
    }
    else {
        current.list<-create.adj(hash.users[u])
        current.el<-current.list$adj.list
        if(is.na(current.el)) {
            print(paste(hash.users[u]), ``has a protected account--ignoring'')
        }
        else {
            hash.el<-rbind(hash.el,current.el)
        }
    }
}
```

Build the network

```
# We now build the network, but we have to make sure the initial seed is not protected
seed.user<-1
while(protected(getUser(hash.users[seed.user]))) {seed.user<-seed.user+1}
# Now create the edgelists for all uses
for(u in seed.user:length(hash.users)) {
    if(u==seed.user) {
        hash.list<-create.adj(hash.users[u])
        hash.el<-hash.list$adj.list
    }
    else {
        current.list<-create.adj(hash.users[u])
        current.el<-current.list$adj.list
        if(is.na(current.el)) {
            print(paste(hash.users[u]), ``has a protected account--ignoring'')
        }
        else {
            hash.el<-rbind(hash.el,current.el)
        }
    }
}
```

Build the network

```
# We now build the network, but we have to make sure the initial seed is not protected
seed.user<-1
while(protected(getUser(hash.users[seed.user]))) {seed.user<-seed.user+1}
# Now create the edgelists for all uses
for(u in seed.user:length(hash.users)) {
    if(u==seed.user) {
        hash.list<-create.adj(hash.users[u])
        hash.el<-hash.list$adj.list
    }
    else {
        current.list<-create.adj(hash.users[u])
        current.el<-current.list$adj.list
        if(is.na(current.el)) {
            print(paste(hash.users[u]), ``has a protected account--ignoring'')
        }
        else {
            hash.el<-rbind(hash.el,current.el)
        }
    }
}
```

Build the network

```
# We now build the network, but we have to make sure the initial seed is not protected
seed.user<-1
while(protected(getUser(hash.users[seed.user]))) {seed.user<-seed.user+1}
# Now create the edgelists for all uses
for(u in seed.user:length(hash.users)) {
    if(u==seed.user) {
        hash.list<-create.adj(hash.users[u])
        hash.el<-hash.list$adj.list
    }
    else {
        current.list<-create.adj(hash.users[u])
        current.el<-current.list$adj.list
        if(is.na(current.el)) {
            print(paste(hash.users[u]), ``has a protected account--ignoring'')
        }
        else {
            hash.el<-rbind(hash.el,current.el)
        }
    }
}
```

Build the network

```
# We now build the network, but we have to make sure the initial seed is not protected
seed.user<-1
while(protected(getUser(hash.users[seed.user]))) {seed.user<-seed.user+1}
# Now create the edgelists for all uses
for(u in seed.user:length(hash.users)) {
    if(u==seed.user) {
        hash.list<-create.adj(hash.users[u])
        hash.el<-hash.list$adj.list
    }
    else {
        current.list<-create.adj(hash.users[u])
        current.el<-current.list$adj.list
        if(is.na(current.el)) {
            print(paste(hash.users[u]), ``has a protected account--ignoring'')
        }
        else {
            hash.el<-rbind(hash.el,current.el)
        }
    }
}
```

Live demo

~~Live~~Pre-loaded demonstration time!
#user2010

All code available at `http://www.drewconway.com/zia`