SocialMediaLab Tutorial - Data Collection

Robert Ackland & Timothy Graham

23 June 2017

Introduction

This is a tutorial for the *SocialMediaLab* R package.

SocialMediaLab was created by Tim Graham (who is also the maintainer of the package) and Robert Ackland. A major contribution was also made by Chung-hong Chan. For more information, see SocialMediaLab page on the VOSON website.

SocialMediaLab enables users to collect social media data and create different kinds of networks for analysis. It is a 'Swiss army knife' for this kind research, enabling a swift work flow from concept to data to fully-fledged network, ready for SNA and other analysis. It can handle large datasets and create very large networks, upwards of a million or more nodes (depending on your computer's resources!). The following data sources are currently supported:

1. Twitter
2. Facebook
3. YouTube

Installation and setup

First we need ensure that the *SocialMediaLab* package is installed and loaded.

We also want to install the *magrittr* package, so we can simplify the work flow by using 'verb' functions that pipe together. We will also be using the *igraph* package for network analysis.

The following commands will check if the packages are installed and install them as necessary, then load them.

```
if (!"SocialMediaLab" %in% installed.packages()) install.packages("SocialMediaLab")
require(SocialMediaLab)

if (!"magrittr" %in% installed.packages()) install.packages("magrittr")
require(magrittr)

if (!"igraph" %in% installed.packages()) install.packages("igraph")
require(igraph)
```

You will also need to get API access for each data source (e.g. Facebook). You will **not be able to collect any data until you have acquired API credentials**. Step-by-step instructions for obtaining API access are available from the SocialMediaLab page on the VOSON website.

Data collection

In this section, we will run through the collection of data from the three data sources.

Before starting the data collection, make sure you have R set to the correct working directory (this is where data files and network maps will be saved). In RStudio you can do this using Session->Set Working

Directory->To Source File Location. The underlying R commands are something like (make sure you use a directory that exists):

```
> getwd()
> setwd("~/SocialMediaLab/tutorial/")
```

Facebook

In *SocialMediaLab*, the process of authentication, data collection, and creating social networks can be expressed with the 3 verb functions: `Authenticate()`, `Collect()`, and `Create()`. This simplified workflow exploits the pipe interface of the *Magrittr* package, and provides better handling of API authentication between R sessions.

What we are doing is "piping" the data forward using the `%>%` operator, in a kind of functional programming approach. It means we can pipe together all the different elements of the work flow in a quick and easy manner.

This also provides the ability to save and load authentication tokens, so we don't have to keep authenticating with APIs between sessions. Obviously, this opens up possibilities for automation and data mining projects.

Make sure we have our `appID` and `appSecret` values defined:

```
appID <- "xxxx"
appSecret <- "xxxx"
```

We will collect one month of activity from the Stop Coal Seam Gas Blue Mountains Facebook fan page.

This will collect all the *posts* posted between the `rangeFrom` and `rangeTo` dates, including all *comments* and *likes*, and other associated data including usernames, timestamps for comments, etc. Note: the date format is YYYY-MM-DD.

We will be using this data to create a bimodal network. This graph object is bimodal because edges represent relationships between nodes of two different types. For example, in our bimodal Facebook network, nodes represent Facebook *users* or Facebook *posts*, and edges represent whether a user has commented or 'liked' a post. Edges are directed and weighted (e.g. if user i has commented n times on post j, then the weight of this directed edge equals n).

Note: for Facebook, *SocialMediaLab* currently only supports creating *bimodal* and *dynamic* networks. More types of networks will be implemented soon.

```
g_bimodal_facebook_csg <- Authenticate("Facebook",
  appID = appID, appSecret = appSecret) %>%
  SaveCredential("FBCredential.RDS") %>%
  Collect(pageName="StopCoalSeamGasBlueMountains", rangeFrom="2015-06-23",
  rangeTo="2015-07-23", writeToFile=TRUE) %>%
  Create("Bimodal")
```

Note that you will receive a message "Copy and paste into Site URL on Facebook App Settings: http://localhost:1410 When done, press any key to continue..." – just press enter and your browser will open up so you can login to Facebook. Once you've successfully logged into Facebook, you should receive the message (in the browser): "Authentication complete. Please close this page and return to R."

The Magrittr pipe approach used in this example means that we only end up with the *final graph object* (in the global environment). To ensure we retain the data that are collected, the argument `writeToFile=TRUE` is used. This writes the data collected using `Collect()` function to a local CSV file before it is piped through

to the network generation function `Create()`. We can then read it in as a dataframe (see code snippet below).

Note: an alternative approach (if you just want the data, not the network) is to remove the `Create()` function from the pipeline, meaning that the `Collect()` function is the final value returned (i.e. the data, which can later be piped through to `Network()` if we want to create network objects).

```
# make sure you change the filename!:
myCSG_data <- read.csv("2015-06-23_to_2015-07-23_StopCoalSeamGasBlueMountains_FacebookData.csv")
View(myCSG_data)
```

This means we end up with two objects for further analysis, a graph object `g_bimodal_facebook_csg`, and a dataframe object `myCSG_data`.

We can now examine the description of our network:

```
g_bimodal_facebook_csg
```

Before plotting the graph, change the node color such that Posts are red, while Users are the default color (blue).

```
V(g_bimodal_facebook_csg)$color <- ifelse(V(g_bimodal_facebook_csg)$type == "Post", "red", "blue")
```

We can see the network with the following:

```
plot(g_bimodal_facebook_csg)
```

In RStudio, the plot pane is generally too small and so an improvement is via opening an X11 graphics driver (only on machines with access to an X server):

```
x11()
plot(g_bimodal_facebook_csg)
```

The following set of commands prints the plot (with some plot options to improve the visualisation) to file:

```
png("g_bimodal_facebook_csg.png", width=800, height=700)
plot(g_bimodal_facebook_csg,vertex.shape="none",edge.width=1.5,edge.curved = .5,edge.arrow.size=0.5,ver
dev.off()
```

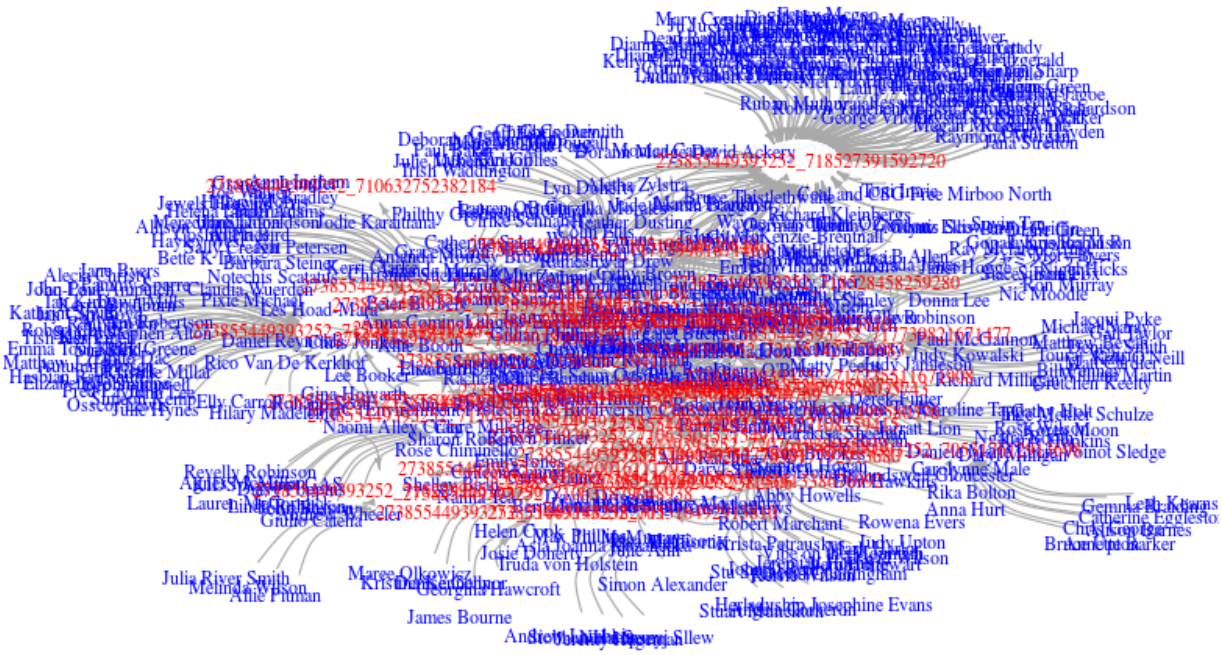
We can save the graph object as a graphml file (so can be visualised using software such as Gephi) or for later use in R.

```
write.graph(g_bimodal_facebook_csg,
  "g_bimodal_facebook_csg.graphml",format="graphml")
```

This can be read back into R as follows (note: to do this, you may have to compile R with XML support):

```
g <- read.graph("g_bimodal_facebook_csg.graphml", format="graphml")
```

This results in the following (your newtwork will look different to this, since it was collected at a different time):



Twitter

It is currently possible to create 3 different types of networks using Twitter data collected with `SocialMediaLab`. These are (1) *actor* networks; (2) *bimodal* networks; and (3) *semantic* networks. In this tutorial we will focus just on *actor* networks.

First, define the API credentials. Due to the Twitter API specifications, it is not possible to save authentication token between sessions. The `Authenticate()` function is called only for its side effect, which provides access to the Twitter API for the current session.

```
# REPLACE WITH YOUR API KEY
myapikey <- "xxxx"
# REPLACE WITH YOUR API SECRET
myapisecret <- "xxxx"
# REPLACE WITH YOUR ACCESS TOKEN
myaccesstoken <- "xxxx"
# REPLACE WITH YOUR ACCESS TOKEN SECRET
myaccesstokensecret <- "xxxx"
```

While it is possible to create a network with one command, here we will `Collect()` the data, but not pipe it directly through to `Network()` straight away. This means we can reuse the data multiple times to create different kinds of networks for analysis.

We will collect 150 recent tweets that have used the #auspol hashtag. This is the dominant hashtag for Australian politics.

```
myTwitterData <- Authenticate("twitter", apiKey=myapikey,
                              apiSecret=myapisecret,
                              accessToken=myaccesstoken,
                              accessTokenSecret=myaccesstokensecret) %>%
Collect(searchTerm="#auspol", numTweets=150,
        writeToFile=TRUE,verbose=TRUE)
```

Note: if you are getting the error `Error in check_twitter_oauth( )`, please find a solution here.

We can have a quick look at the data we just collected:

```
View(myTwitterData)
```

Note the class of the dataframe, which lets `SocialMediaLab` know that this is an object of class `dataSource`, which we can then pass to the `Create()` function to generate different kinds of networks:

```
class(myTwitterData)
```

If you find that you are encountering errors possibly related to the text of the tweets, you can try converting the tweet text to UTF-8 character encoding:

```
myTwitterData$text <- iconv(myTwitterData$text, to = 'utf-8')
```

Mac users only may also wish to try the following if they are encountering errors that may be due to character encoding issues:

```
#myTwitterData$text <- iconv(myTwitterData$text, to = 'utf-8-mac')
```

Now, we will create an *actor* network. In this actor network, edges represent interactions between Twitter users. An interaction is defined as a 'mention' or 'reply' or 'retweet' from user i to user j, given 'tweet' m. In a nutshell, a Twitter actor network shows us who is interacting with who in relation to a particular hashtag or search term.

```
g_twitter_actor <- myTwitterData %>% Create("Actor")
```

We now have an *igraph* graph object, and can analyse it using the same techniques we used in the "Divided They Blog" exercise.

We can now examine the description of our network:

```
g_twitter_actor
```

Next, we can visualise the network by plotting it directly in R:

```
png("g_twitter_actor.png", width=800, height=700)
plot(g_twitter_actor,vertex.shape="none",edge.width=1.5,edge.curved = .5,edge.arrow.size=0.5,asp=9/16,ma
dev.off()
```

This results in the following (your newtwork will look different to this, since it was collected at a different time):

Finally, save the graph object as a graphml file:

```
write.graph(g_twitter_actor,
  "g_twitter_actor.graphml",format="graphml")
```

YouTube

As with the other data sources, the first thing we do is set up the API credentials:

```
apiKey <- "xxxx"
```

Next, we specify which videos we want to collect data from, using a character vector specifying one or more YouTube video IDs. For example, if the video URL is 'https://www.youtube.com/watch?v=Xfo0hpVrtrs', then use `videoIDs="Xfo0hpVrtrs"`.

For this example, we will collect data from an Australian anti-fracking video featuring 70s pop star Leo Sayer (!):

```
videoIDs <- c("Xfo0hpVrtrs")
```

The workflow is fairly straightforward - we just pipe together the 'verb' functions. A couple of comments.

1. By default, all the available comments are collected. If desired, the 'maxComments' argument can be used to limit the number of comments (but as noted in the documentation, this is not always perfect, due to the YouTube API specifications).
2. Often, we will wish to retain the comment text for further analysis. There are two approaches (as discussed previously). First option is to leave out `Create()` function from the pipeline, so we are just creating a dataframe object with our data (which we can later pipe through to `Create()` an actor network). The second option, which we use in this example, is to specify `writeToFile=TRUE`, so we write the data to disk before piping it through `Create()` the network.

```
g_youtube_actor <- Authenticate("youtube", apiKey= apiKey) %>%
  Collect(videoIDs = videoIDs, writeToFile=TRUE) %>%
  Create("Actor")
```

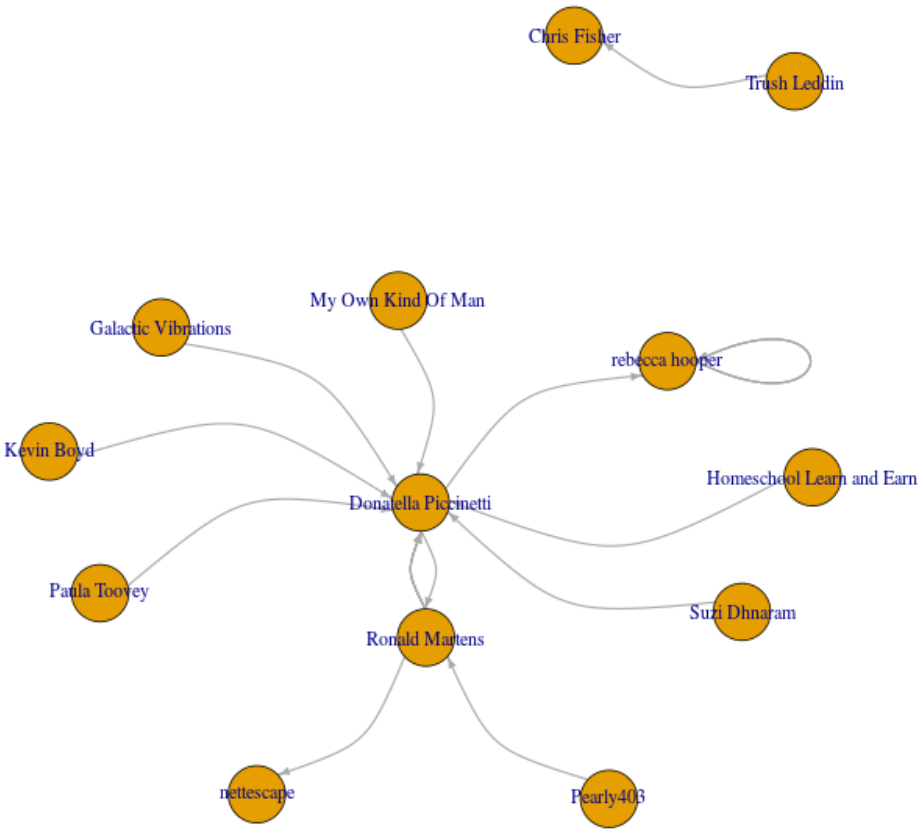
A description of the network:

```
g_youtube_actor
```

A plot of the network:

```
png("youtube_fracking.png", width=800, height=700)
plot(g_youtube_actor,edge.width=1.5,edge.curved=.5,edge.arrow.size=0.5)
dev.off()
```

This results in the following:



Save the graph object as a graphml file:

```
write.graph(g_youtube_actor,
  "g_youtube_actor.graphml",format="graphml")
```

Read in the YouTube data that we saved to disk, for example:

```
# make sure you change the filename:
myYouTubeData <- read.csv("Jun_26_03_12_16_2017_AEST_YoutubeData.csv")
View(myYouTubeData)
```