Lab 1: Visualising Social Media Networks

Social Media and Web Analytics @ TiSEM

Last updated: 29 April, 2023

Motivation

Digital Social Networks - the connections between users - are the basis of all social media platforms. To build an understanding of social media, one could reasonably argue we first need to understand the connections between users.

In this tutorial you will learn how to construct networks of Twitter users based on their retweet behaviour, who a user mentions in a tweet or who a user replies to. Using an existing data set you will learn how to visualise a Twitter network based on replies to an original tweet.

Learning Goals

By the end of this tutorial you will be able to:

- Construct an edge list that connects Twitter users based on mentions, replies and/or retweets
- Plot networks of Twitter users using `tidygraph` and `ggraph`

Instructions to Students

These tutorials are **not graded**, but we encourage you to invest time and effort into working through them from start to finish. Add your solutions to the `lab-01_answer.Rmd` file as you work through the exercises so that you have a record of the work you have done.

Obtain a copy of the answer file using Git. To clone a copy of this repository to your own PC, use the following command:

```
$ git clone https://github.com/tisem-digital-marketing/smwa-lab-01.git
```

Once you have your copy, open the answer document in RStudio as an RStudio project and work through the questions.

The goal of the tutorials is to explore how to "do" the technical side of social media analytics. Use this as an opportunity to push your limits and develop new skills. When you are uncertain or do not know what to do next - ask questions of your peers and the instructors on the classes Slack channel `#lab01-discussion`.

Plotting Social Media Networks

You will need to use the following `R` libraries throughout this exercise:

```
library(readr)
library(tidygraph)
library(ggraph)
library(dplyr)
library(tidyr)
library(tibble)
```

In this exercise you will work with some existing Twitter data. The data you will use is a collection of tweets that all have the hashtag `#rstats` in their text.

The data are collected from 2018, and can be downloaded from the internet:

```
url <- "https://bit.ly/3r8Gu4M"
# where to save data
out_file <- "data/rstats_tweets.rds"
# download it!
download.file(url, destfile = out_file, mode = "wb")
```

The data that you downloaded are an `.rds` file, so you can load them with the `read_rds` function from the `readr` library:

```
tweets <- read_rds(out_file)
```

Your goal will be to construct a network graph that visualizes the connections between Twitter users. For this exercise you are interested in connections between Twitter users who reply to each others tweets. That is, two users are connected if user A has replied to user B's tweet or vice versa.

Before you work through the guided exercise, we recommend that you take some time to look at the data and understand it's basic structure. There are a lot of column names, and you will want to understand what is in them.

Now, let's begin the analysis:

1. Create a new data set that only includes tweets that contain replies:

```
tweets_replies <- # YOUR CODE HERE
```

solution

```
tweets_replies <-
  tweets %>%
  drop_na(reply_to_screen_name)
```

2. Further reduce the size of the data by dropping the columns you will not need. Your new data set should only include the columns named `screen_name` and `reply_to_screen_name`. Rename these columns to `from` and `to`.

```
tweets_replies <- # YOUR CODE HERE
```

solution

```
tweets_replies <-
  tweets_replies %>%
  select(from = screen_name, to = reply_to_screen_name)
```

3. When a user on Twitter writes a long series of tweets about the same topic, they often connect multiple tweets together by replying to their own previous tweet to chain their posts together. Remove these replies from the data.

```
edgelist <- # YOUR CODE HERE
```

solution

```
edgelist <-
  tweets_replies %>%
  filter(from != to)
```

4. Now, you are going to trim down the size of the edge list. You will do this mainly so that your computer won't freeze when it comes time to plot the network.[1] Proceed in two steps:

(a) Create a data set that counts the number of times a user replies to anyone in the data. Keep only users who have replied more than 50 times.

```
interactions_sent <- # YOUR CODE HERE
```

solution

```
interactions_sent <-
  edgelist %>%
  count(from) %>%
  arrange(desc(n)) %>%  # this step is unnecessary
  filter(n > 50)
```

(b) Update your the edge list so that only users who have engaged in at least 50 replies are included.

```
edgelist <- # YOUR CODE HERE
```

solution

```
edgelist <-
  edgelist %>%
  # the first of the two lines below filters to include only repliers
  # in the interactions_sent data frame
  # the second line does the same, for reply recievers
  filter(from %in% interactions_sent$from,
         to %in% interactions_sent$from)
```

5. Convert your `data.frame` containing all the edges to a tidygraph object.

```
tg <- # YOUR CODE HERE
```

solution

```
tg <- as_tbl_graph(edgelist)
```

6. Plot the network. Use the layout `kk` in your solution.

```
# YOUR CODE HERE
```

solution

[1] Plotting large networks can be challenging on your computer's RAM which will lead to it freezing. We're trying to stop this behaviour.

```
  ggraph(tg,
         layout = "kk"
         ) +
  # this adds the points to the graph
  geom_node_point() +
  # this adds the links, or the edges;
  # alpha = .2 makes it so that the lines are partially transparent
  geom_edge_link(alpha = .2) +
  # this last line of code adds a ggplot2 theme suitable for network graphs
  theme_graph()
```

7. Explore different layouts and find the one you think works best visually. You could explore the choice "stress" or any of the following: "dh", "drl", "fr", "gem", "graphopt", "lgl", "mds", "sugiyama", "bipartite", "star" or "tree".

8. The plot you produced weighted the edges by the frequency in which two nodes had replied to each other (it does this implicitly because the same edge occurred many times in the edge list). Prevent this from happening by adjusting the edgelist to only contain distinct entries and replotting the graph. You will have to re-run parts of you code to produce the new graph.

```
# YOUR CODE HERE
```

solution

```
edgelist <-
  edgelist %>%
  dplyr::distinct() %>%
  # the first of the two lines below filters to include only senders
  # in the interactions_sent data frame
  # the second line does the same, for receivers
  filter(from %in% interactions_sent$from,
         to %in% interactions_sent$from)

tg <- as_tbl_graph(edgelist)

ggraph(tg,
       layout = "kk"
       ) +
  # this adds the points to the graph
  geom_node_point() +
  # this adds the links, or the edges;
  # alpha = .2 makes it so that the lines are partially transparent
  geom_edge_link(alpha = .2) +
  # this last line of code adds a ggplot2 theme suitable for network graphs
  theme_graph()
```

9. You can add color to the plot that you just created. Color (and re-size) the nodes based on their influence as measured by `centrality_authority`.

```
# YOUR CODE HERE
```

solution

```
tg <- tg %>%
# this calculates the centrality of each individual using the built-in
  # centrality_authority() function
  mutate(centrality = centrality_authority())

ggraph(tg,
       layout = "kk"
       ) +
  geom_node_point(
    aes(
      size = centrality,
      color = centrality
      )
    ) +
  # this line colors the points based upon their centrality
  scale_color_continuous(guide = 'legend') +
  geom_edge_link(alpha = 0.05) +
  theme_graph()
```

10. Save the last plot you created as 'rstats-replies.pdf'.

```
# YOUR CODE HERE
```

solution

```
ggsave("rstats-replies.pdf")
```

Bonus Exercise on Plotting

The graphs you have created so far use fairly standard algorithms to plot the data. An alternative that sometimes produces nicer looking plots uses an algorithm called `backbone`. This exercise utilizes `backbone` to plot the data.

You will need to install an additional package:

```
install.packages("graphlayouts")
```

And then load it into your `R` session:

```
library(graphlayouts)
```

1. The `backbone` layout takes a `tidygraph` object and creates a layout. Run the following code to do this (don't worry too much about what this is doing in the background):

```
# assumes your network graph is stored as
# a tidygraph object labelled "tg"
bb <- graphlayouts::layout_as_backbone(tg, keep = 0.4)
E(tg)$col <- FALSE
E(tg)$col[bb$backbone] <- TRUE
```

2. Use the `backbone` layout to create a graph as by extending the following code chunk:

```
# don't change the first line unless your graph
# has a different name than "tg"
ggraph(tg, layout = "manual", x = bb$xy[, 1], y = bb$xy[, 2]) +
    geom_node_point(YOURCODE) +
```

```
    geom_edge_link(YOURCODE) +
    YOURCODE
```

3. Try different values of the `keep` argument in (1) and plot the figure again. Do you notice any differences as you do?

License

Suggested Citation

Deer, Lachlan. 2023. Social Media and Web Analytics: Lab 1 - Visualizing Social Media Networks. Tilburg University. url = "https://github.com/tisem-digital-marketing/smwa-lab-01"