Watts & Strogatz model

duw

11/06/2018

The goal of this assignment is to create a social network from your free recall data, to plot it and to evaluate it for smallworldness and centrality.

Overview

This assignment contains of 3 steps.

1. Create a graph from the responses.
2. Plot the graph using `ggraph`.
3. Analyze it using `igraph`.

Step I - Create social network

1. First, download the data using this **link** and store it inside your project. Then load the data using the following command (you probably don't need the `..`). Inspect the data. You will see that the object is composed of a list of vectors, with every vector representing the responses of one person. Which person the responses belong to is coded in the list's names. E.g., `free_recall$"Zana Hightower"` would give you the responses of Zana Hightower.

```
# load data
free_recall = readRDS("../1_data/psychonet_responses.RDS")
free_recall$"Zana Hightower"
```

```
##  [1] "Juliana Lemus"    "Phylicia Belcher" "Cassy Martino"
##  [4] "Sigrid March"     "Reena Place"      "Lourie Henke"
##  [7] "Gerry Dolan"      "Velva Burley"     "Tona Timm"
## [10] "Jarvis Chapin"    "Deandre Talbert"  "Alysha Harwood"
## [13] "Lory Ralston"     "Luciano Aiken"    "Dee Bartholomew"
## [16] "Sharie Gable"     "Maudie Arroyo"    "Maisha Van"
```

2. Extract all of the respondents names using `names(free_recall)` and the unique responses using `unique(unlist(free_recall))`. Store these in objects named `respondents` and `responses` and create a third one containing the unique names across both, using `unique(c(respondents, responses))`, and name it `persons`.

3. Now create an adjacency matrix with enough rows and columns to store the edges between individuals using `matrix(0, ncol = XX, nrow = XX)` (tipp: what is the `length()` of `persons`?) and name it `social_network`. Assign the matrix' `rownames()` and `colnames()` to be the names contained in `persons` (e.g., `rownames(XX) <- XX`).

4. Now comes the somewhat difficult part. Iterate over the free recall list and include an edge, if respondent `i` produced response `j`, i.e., set the cell `i, j` to `1`. Do this using an outer-loop iterating over the respondents and an inner-loop iterating over the respondents' responses. For every respondent this means that you need to pull the respondent's responses and then iterate over those. Note that objects of type `list` and `matrix` can also be accessed using names. Here, we can make use of this by iterating directly over the names rather than their indices. See below.

```
# fill social network
for(i in respondents){

  # HERE EXTRACT RESPONSES OF RESPONDENT i
  responses_i = free_recall[[XX]]

  # loop over responses
  for(j in  responses_i){

    # add edges
    social_network[XX, YY] = social_network[YY, XX] = ZZ

  }
}
```

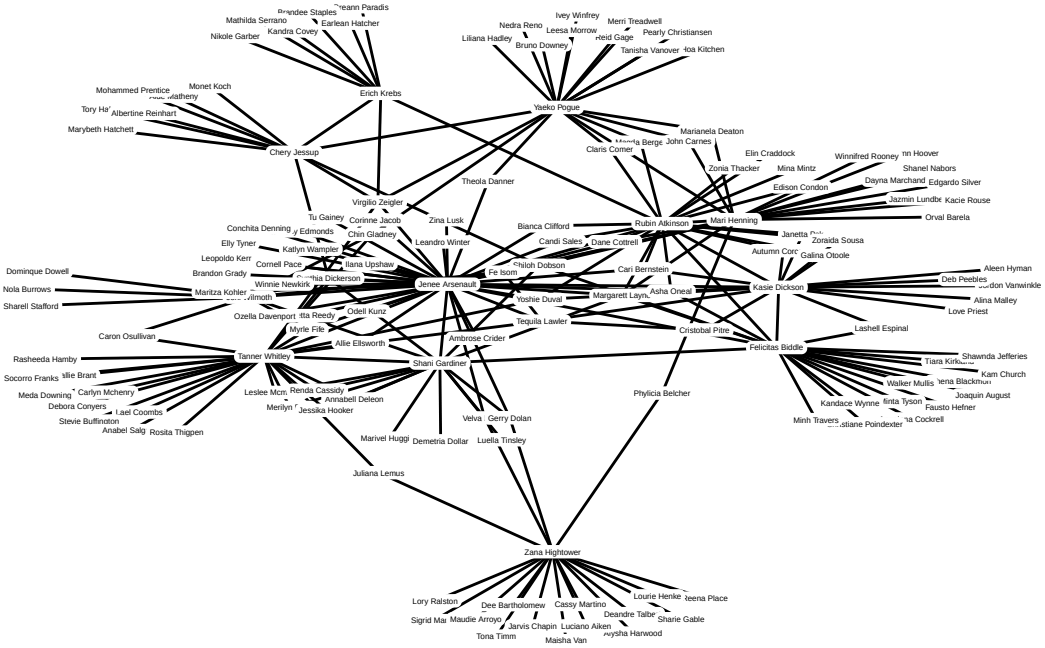
Step II - Plot social network

1. The next step is to plot the network. A lot of plotting can be done using `igraph`, but `ggraph` clearly creates nicer plots (and has nicer syntax). Install the package using `install.packages('ggraph')` and check out the **Intro**. Then plot the network using the code below. If the graph and the labels appear too small, increase the numbers in `geom_node_label()`.

```
require(igraph)
require(ggraph)

# create social graph
social_graph <- graph_from_adjacency_matrix(social_network, mode = 'undirected')

# plot
ggraph(social_graph) +
  geom_edge_link() +
  geom_node_label(aes(label = V(social_graph)$name),
                  repel = FALSE,
                  label.size = unit(0.15, "lines"),
                  label.padding = unit(.1, "lines"),
                  size = 1.2) +
  theme_graph(base_family = "Roboto Condensed")
```



Step III - Analyze social network

1. Is your social network a *small world*? To evaluate this, calculate its clustering coefficient (using `transitivity(XX, type = 'localaverage')`) and average shortest path length (using `average.path.length(XX)`). What do you think, is it small world?

2. Identify central individuals. Use the functions `centr_degree()`, `centr_clo`, and `centr_betw` to calculate the centrality with regard to the respective definiton for every person (i.e., node). Then identify the, respectively, most central person using `persons[which.min(XX)]`. Note, each of the three functions returns a list. This means that you first need to extract the vector containing the centrality values (usually called `res`).

3. BONUS: If you are interested in finding communities you can play around with `cluster_louvain()` and `cluster_optimal()` from the igraph packages.