PsychoNet pt.2

duw

11/12/2018

The goal of this assignment is apply random walks to your social network to simulate the spread of information and to appreciate problems associated with measuring social distance.

Overview

This assignment consists of 2 steps.

1. Build random walker.
2. Study random walker results.

Step I - Create random walker

1. The first step of creating a random walker is to create a function that identifies the neighbors of node `i`. Doing this is relatively straightforward using the adjacency matrix of the social network: A row `i` of the adjacency matrix codes which nodes `j` are connected (represented by `1`) and which are not connected (represented as `0`) to node `i`. That is, all you need to do is to identify the locations where the row has value `1`. This can be done using the `which()` function. Specifically, `which(row == 1)` will return the indices at which a `row` object (containing a row of the adjacency matrix) is equal to `1`. Use this to create a simple function called `get_neighbors()` that returns all neighbors for any node `i` using the template below.

```
# define get neighbors function
get_neighbors <- function(index, network){

  # get neighbors
  row <- XX
  neighbors <- XX

  # return
  neighbors
  }
```

```
# define get neighbors function
get_neighbors <- function(index, network){

  # get neighbors
  row <- network[index, ]
  neighbors <- which(row == 1)

  # return
  neighbors
  }
```

2. Now make sure that the function works. This should be results when using the function to retrieve the neighbors of row 6 ('Zana Hightower').

```
##    Juliana Lemus     Velva Burley      Gerry Dolan Phylicia Belcher 
##               57               79               81               86 
##    Cassy Martino     Sigrid March      Reena Place     Lourie Henke 
##               87               88               89               90 
##        Tona Timm    Jarvis Chapin  Deandre Talbert   Alysha Harwood 
##               91               92               93               94 
##     Lory Ralston    Luciano Aiken  Dee Bartholomew     Sharie Gable 
##               95               96               97               98 
##    Maudie Arroyo        Maisha Van 
##               99              100
```

3. The next step is to randomly sample a neighbor. You already have the means to select the set of neighbors. To sample a random neighbor means to pick one neighbor from the set of neighbors by random. This can be done using the `sample()`-function. The `sample()`-function expects the set from which to choose from as the first argument and the number of to-be-chosen elements as the second argument. I.e., `sample(neighbors, 1)` gives you one randomly chosen neighbor. Try it out!

```
neighbors <- get_neighbors('Zana Hightower', social_network)
sample(neighbors, 1)
```

```
## Lory Ralston 
##           95
```

4. Ok, now put `sample(neighbors, 1)` and `get_neighbors()` together into a new function named `get_neighbor()` (singular).

```
# define get neighbors function
get_neighbor <- function(index, network){

  # CODE HERE

  }
```

```
# define get neighbors function
get_neighbor <- function(index, network){

  # get neighbors
  row <- network[index, ]
  neighbors <- which(row == 1)

  # return
  sample(neighbors, 1)
  }
```

5. Using the `get_neighbor()`-function, you can now set-up the random walker function that repeatedly applies the `get_neighbor()` function to traverse the network. This is done by using at every step the newly drawn neighbor as the new index. Begin writing a function called `random_walk` that takes three arguments, the `index`, the `network`, and the maximum number of steps `n_steps`.

```
# define get neighbors function
random_walker <- function(index, network, n_steps){
```

```
  # CODE HERE

  }
```

6. Inside the function, now create a loop that repeats the `get_neighbor()` for changing inidices. That is `get_neighbor()` is first exectued for `index`, then for the node returned by `get_neighbor()` and so on until `n_steps` have been performed. While iterating over `1:n_steps` store the visited nodes in vector and return it at the end of the function.

```
# define random walker function
random_walker <- function(index, network, n_steps){

  # set start index
  current_node <- index

  # set container
  nodes = c()

  # loop until n_steps
  for(i in 1:n_steps){
    current_node = get_neighbor(current_node, network)
    nodes[i] = current_node
    }

  # return
  nodes
  }
```

Step II - Study random walker

Feel free to choose any of the tasks (or all).

A. Which node is most visited by random walks? Try it out: let the random walker run (for a long time, e.g., `n_steps > 1000`) and evaluate how often every node occurrs. Use the `sort(table())` function. Which nodes has the most visits? Does it matter where the random walker was started? Compare the results to the centrality measure results from the previous assignment.

```
# start node
i = which(rownames(social_network) == 'Jenee Arsenault')

# count visits
result = random_walker(i, social_network, 10000)
tab = sort(table(result))

# extract most visited node
rownames(social_network)[as.numeric(names(tab))[1]]
```

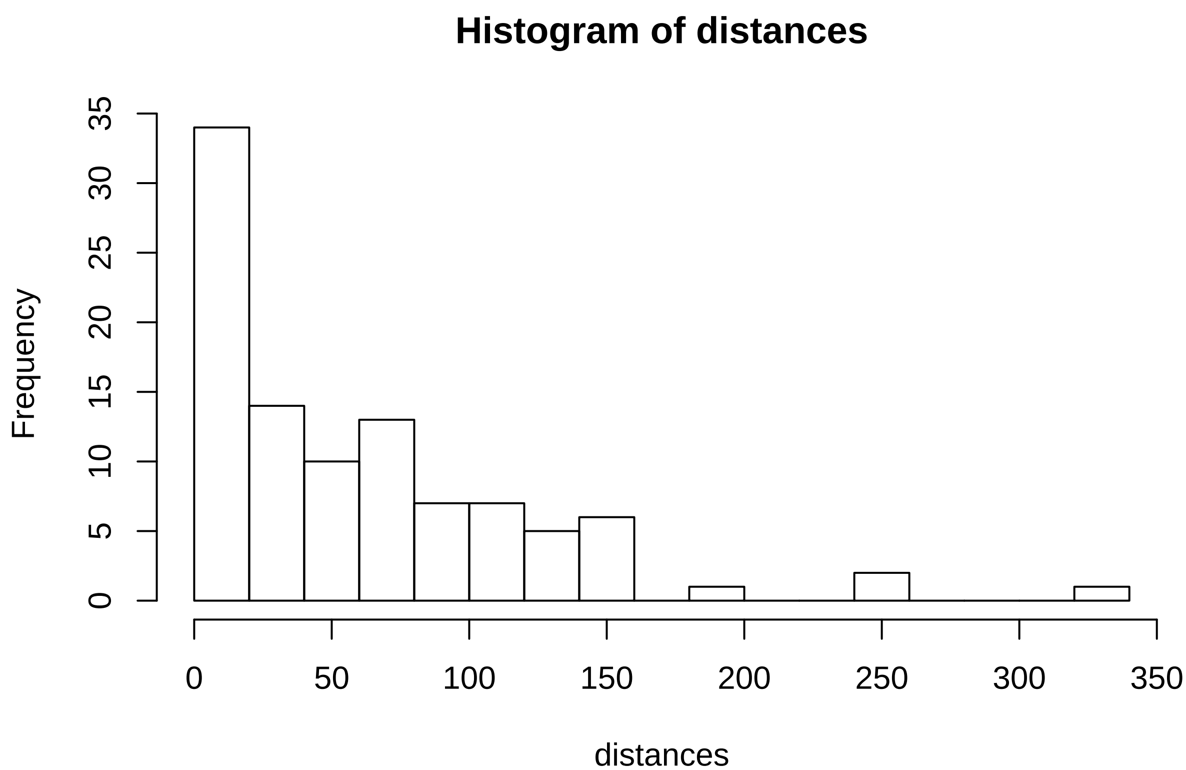
```
## [1] "Jarvis Chapin"
```

B. How many steps are needed, on average (i.e., not the shortest distance), to get from point i to point j? Try it out: (a) choose two nodes, (b) choose one of them to be the start node, (c) let the random walker run for a large number of steps, and (d) evaluate how many steps it took to get to the other chosen node. Evaluate this using `min(which(sequence == j))` which will give the first time at which the `j`-th node was visited in the sequence of nodes (produced by the random walk). E.g., `min(which(random_walker(1, social_network, 1000) == 2))`. Do the numbers match your expectations? What is the average number of steps needed to connect two distant nodes?

```
# get start nodes
i = which(rownames(social_network) == 'Jenee Arsenault')
j = which(rownames(social_network) == 'Yaeko Pogue')

# get distances
distances = c()
for(k in 1:100){
  distances[k] = min(which(random_walker(i, social_network, 1000) == j))
  }

# show distances
hist(distances, breaks = 20)
```



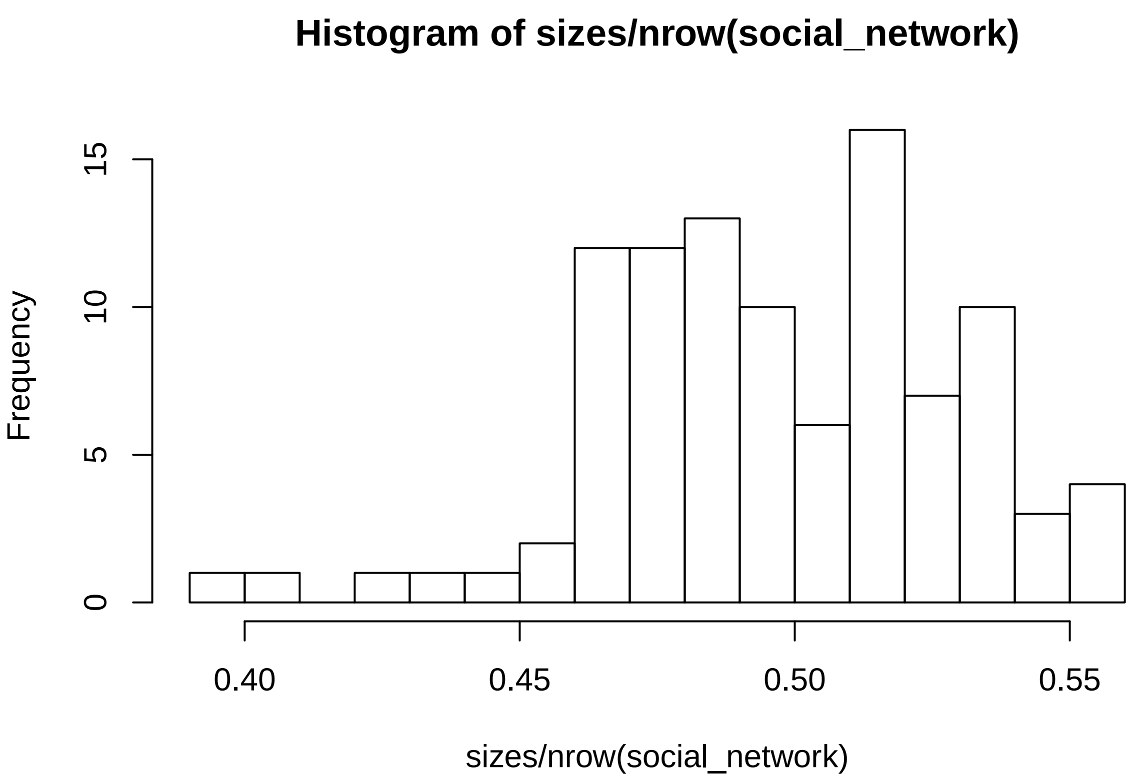
C. How long does it take for the random walk to cover 50% of the nodes depending on the start node. Try it out: choose a start node and then let it run for a certain number of steps. Each time count how many different nodes have been covered using `length(unique(sequence))`. Play around how many steps does it roughly need to cover half the network, i.e., 74 nodes. Does it matter where you start?

```
# start node and number of steps
i = which(rownames(social_network) == 'Jenee Arsenault')
n_steps = 250

# get number of visited nodes
sizes = c()
for(k in 1:100){
```

```
  sizes[k] = length(unique(random_walker(i, social_network, n_steps)))
  }

# show proportion of visited nodes
hist(sizes / nrow(social_network), breaks = 20)
```



D. (Advanced) Viewing task B as a model of information spreading or communication, the implementation essentially assumed that every person merely talked to a single other person. In reality, however, people talk to to more than one person, possibly even their entire neighborhood? Try to program code that evaluates how the number of visited nodes changes as a function of how many nodes each node communicates whith. Be careful, this problem can easily become very computationally intensive (i.e., start with small numbers).

```
# start node
talkers = which(rownames(social_network) == 'Jenee Arsenault')

# get spreading function
spreading_activation = function(start, n_comm, n_steps){

  # setup
  talkers = start
  visited = c()

  # simulate
  for(i in 1:n_steps){
    new_talkers = c()
    for(j in 1:length(talkers)){
      neighbors = get_neighbors(talkers[j], social_network)
      if(length(neighbors) > n_comm) neighbors = sample(neighbors, n_comm)
      visited = unique(c(visited, neighbors))
```

```r
      new_talkers = unique(c(new_talkers, neighbors))
      }
    talkers = new_talkers
    }
  visited
  }

# get results
length(spreading_activation(1, 4, 4))
```

```
## [1] 43
```