Lightning Network R Documentation

Susheel Palakurthi / Julius Wu / Vic Dhand / Zack Dupont

April 5, 2018

Installation and implementation of packages

```
install.packages("memoise")
install.packages("sqldf")
install.packages("igraph")
install.packages("network")
install.packages("visNetwork")
install.packages("networkD3")

library(memoise)
library(sqldf)
library(igraph)
library(network)
library(visNetwork)
library(networkD3)
```

These are the packages that are required in the project. The functions within these packages will be used to carry out the tasks.

Data Background and Management

The data is sourced from shabang.io. This website contains data from public transactions made on the lightning network for Bitcoin (BTC). The data is collected every minutes so our project only contains a snippet of the current data, as mentioned by shabang.io, "Every 15 minutes, we take a snapshot of the network and log the results below."

Data was collected March 19 2018 at 4:38 PM.

Original data: nodes, channels, stats. nodes: information about each node with nodeid and alias channels: the information that make up the connections of transactions between each node stats: extraneous data that isn't dealt with for this project

At the time the data was collected, there were: 842 nodes 4778 channels 514312 blockheight

All three files (nodes, channels, stats) were downloaded as .json files from shabang.io. They were then converted into .csv files for reading into Microsoft Excel and R.

The main file of interest is channels. The problem with this dataset is the size of the data. It originally contained 4779 connections (edges). The only solution is to cut down the size of the data. Since there were noc olumns in the channels data to subset the data, we subsetted the data from the 843 nodes.

At first thought, we could've cut the nodes into hald or quarters but by doing so, we'd lose a large amount of information. Instead, we decided to subset the data with available information. From the nodes data, the addresses/0/address contained information on IP addresses. We used Excel and eliminated missing, encrypted, and extraneous data. These remaining IP addresses were then converted into addresses using an online geolocator containing information on city, area, and country. A Google Chrome extension was used to convert these addresses into longitude and latitude coordinates.

From 843 nodes, the nodes.csv now contained 381 nodes.

In order to subset the channels data for analysis and visualization, both channels and nodes data had to be combined and all compelte rows would remain. Since the channels data has information on source and destination, it'll now have information on the location of each source and destination of edges. All columns that don't match from the subsetted nodes data is removed though some destination information haven't been completely removed.

Basic Data Analysis

```
sioNodes <- read.csv("nodes.csv", header = TRUE)
colnames(sioNodes)
```

```
##  [1] "nodeid"              "alias"               "color"
##  [4] "last_timestamp"      "addresses.0.type"    "addresses.0.address"
##  [7] "addresses.0.port"    "addresses.1.type"    "addresses.1.address"
## [10] "addresses.1.port"
```

```
sioChannels <- read.csv("channels.csv", header = TRUE)
colnames(sioChannels)
```

```
##  [1] "source"                "destination"
##  [3] "short_channel_id"      "flags"
##  [5] "active"                "public"
##  [7] "last_update"           "base_fee_millisatoshi"
##  [9] "fee_per_millionth"     "delay"
```

```
sioStats <- read.csv("stats.csv", header = TRUE)
colnames(sioStats)
```

```
## [1] "height"                   "lightning_nodes"
## [3] "lightning_channels"       "segwit_percentage"
## [5] "segwit_inputs"            "total_inputs"
## [7] "lightning_channels_funded"
```

When downloading data from shabang.io, the three files contain the columns as followed from the column names (colnames function).

```
nodes <- read.csv("nodesWithLoc.csv", header = TRUE)
```

After the steps to convert the original nodes file to something that can be more subsettable was done, the remaining file is nodesWithLoc.csv

Using https://www.ipligence.com/iplocation, we enter the node IP addresses into the website and convert them into continent, country, region, and city. Then, using the Google Chrome Extension, Geocode Cells: https://chrome.google.com/webstore/detail/geocode-cells/pkocmaboheckpkcbnnlghnfccjjikmfc?hl=en, we convert the addresses into Longitude and Latitude.

```
colnames(nodes)
```

```
## [1] "NodeID"    "alias"     "ip"        "continent" "country"   "region"
## [7] "city"      "Lat"       "Lng"
```

```
head(nodes)
```

```
##                                                                 NodeID
## 1 03b7d8b6bbaf02239277ed32378d50d29840292b61d516d30057b74044992c93dd
## 2 0330c464cb2be97cd4ca5057b192a2be3c775a5f0356aced805769cb8790b879c0
## 3 0267db6fbe76e1fbae50e27529a330837e9e1f4b9e4c7bbfefd7d6a1b3ffe2b245
## 4 026b4f8931fcf87033d0f601ad7e4baa8e93ee74acf313292fd397fd6c27524162
## 5 03efeea8961f376931a390ed9ae62be116abc1a8abaad6d1998efcc11d63e86526
## 6 0207197d1028b8a7edfee28f4e2dc47905333b2c4f9ed40bcd2c7481abe8fb049a
##                         alias              ip     continent
## 1                      ALIBABA 173.212.218.204 North America
## 2              Nick's LND Node      51.6.89.26        Europe
## 3                        OKEX    35.231.55.53 North America
## 4 lightning.nicolas-dorier.com   52.166.90.122 North America
## 5                   enthusiast   70.29.131.217 North America
## 6              lightningpay.me  159.65.251.228 North America
##          country               region       city      Lat          Lng
## 1  United States           New Jersey     Newark 40.73566  -74.1723667
## 2 United Kingdom              England     London 51.50735   -0.1277583
## 3  United States             Michigan  Ann Arbor 42.28083  -83.7430378
## 4  United States             Delaware Wilmington 34.22573  -77.9447102
## 5         Canada              Alberta    Calgary 51.04862 -114.0708459
## 6  United States District Of Columbia Washington 47.75107 -120.7401385
```

The above statement is used to check the column names and the first few (6) rows of the imported nodes file.

Now each IP address now has the respective continent, country, region, city and latitude and longitude information.

```
summary(nodes)
```

```
##                                                                 NodeID
##  0201c0130bff6c44f7467cfc4d1343741521489177c24fd9e6fb346a16026ee978:  1
##  0201d3fb63901ffaaa6ce535c9227fb3dc4646b6276096dd10c8a12e178ac9886f:  1
##  020331a1a81bd8ad2c2ffcd6730bee311c6f457ceca8e8cfcfef0117cebfac348a:  1
##  0203eb5c3ae8d060e5c16795a0d7ebccb98a0fb2d68a6d298e3efce28b803b6778:  1
##  02049355ddbff2082890b26cc71edff6600174f9b6971dfdf560324d3577131d5c:  1
##  020600299b8f383a61781d19f917f355fc712e1e0e58dd4d3e3e1dd4b5ebb38f7a:  1
##  (Other)                                                           :375
```

```
##                    alias                    ip               continent
##  mainnet.yalls.org    :  3   51.15.84.98    :  5   Asia         :  9
##  #HayabusaPOLAND#     :  2   71.68.48.149   :  3   Europe       : 48
##  LIGHTSPEED           :  2   104.198.96.115 :  2   North America:323
##  ThrobbingSausage     :  2   173.212.254.206:  2   South America:  1
##   ???? 20-20.se ????  :  1   173.71.149.66  :  2
##   ???? trimigo.se ????:  1   173.95.130.175 :  2
##  (Other)              :370   (Other)        :365
##           country                      region                    city
##  United States :307   Michigan            :133   Ann Arbor         :102
##  United Kingdom: 17   New Jersey          : 53   Newark            : 32
##  Canada        : 16   Texas               : 25   Harrington Park   : 19
##  Spain         : 15   California          : 21   Rochester         : 19
##  Germany       : 13   England             : 15   Houston           : 16
##  Japan         :  9   District Of Columbia:  9   Sault Sainte Marie: 11
##  (Other)       :  4   (Other)             :125   (Other)           :182
##       Lat              Lng
##  Min.   :-34.03   Min.   :-122.981
##  1st Qu.: 38.60   1st Qu.: -83.743
##  Median : 42.28   Median : -79.383
##  Mean   : 38.02   Mean   : -52.945
##  3rd Qu.: 43.16   3rd Qu.:  -8.721
##  Max.   : 59.40   Max.   : 150.731
##
```

At first glance at this data, we can see that there a multiple occurrences of multiple same-named aliases. This is not a problem because the nodeid is what differentiates the connections. Most of this data contains node information from North America. Asia: 9. Europe: 48. North America: 323. South America: 1. Michigan has the most number of nodes, withi its city, Ann Arbor, having 102 nodes.

```
channels <- read.csv("channelsR.csv", header = TRUE)
```

The original channels data has 4778 rows. Since we don't need any other column in channels.csv other than source, destination, and weight, Excel was used to remove all other columns.

```
sourceC <- sqldf(
  "SELECT *
   FROM channels, nodes
   WHERE nodes.NodeID = channels.source"
)
```

To find the continent, country, region, city, latitude and longitude of each IP address of each source, SQL in R was used to process the data.

```
colnames(sourceC)[5] <- "alias.S"
colnames(sourceC)[6] <- "ip.S"
colnames(sourceC)[7] <- "continent.S"
colnames(sourceC)[8] <- "country.S"
```

```
colnames(sourceC)[9] <- "region"
colnames(sourceC)[10] <- "city.S"
colnames(sourceC)[11] <- "Lat.S"
colnames(sourceC)[12] <- "Lng.S"
```

From this SQL result, the column names were changed to signify the source's alias, ip, continet, country, region, city, longitude and latitude coordinates, signified by the suffix ".S"

```
destinationC <- sqldf(
  "SELECT *
   FROM channels, nodes
   WHERE nodes.NodeID = channels.destination"
)
```

To find the continent, country, region, city, latitude and longitude of each IP address of the destination.

```
colnames(destinationC)[5] <- "alias.D"
colnames(destinationC)[6] <- "ip.D"
colnames(destinationC)[7] <- "continent.D"
colnames(destinationC)[8] <- "country.D"
colnames(destinationC)[9] <- "region.D"
colnames(destinationC)[10] <- "city.D"
colnames(destinationC)[11] <- "Lat.D"
colnames(destinationC)[12] <- "Lng.D"
```

From this SQL result, the column names were changed to signify the destination's alias, ip, continent, country, region, city, longitude and latitude coordinates, signified by the suffix ".D"

```
final <- sqldf(
  "SELECT *
   FROM sourceC, destinationC
   WHERE sourceC.source = destinationC.source AND
  sourceC.destination = destinationC.destination AND
  sourceC.weight = destinationC.weight"
)
```

The rows that were duplicated were eliminated by the source and destination datafiles were combined using multiple conditions.

```
write.csv(final, file = "final.csv")
```

Although this step is not necessary, the final result of the SQL statements can be exported into its own csv file.

There are 812 matching records from source and destination from the nodes to the channels with geolocated addresses. This "final" data frame has all the information of source and destination with geolocated locations.

```
channelsR <- read.csv("newChanneldata.csv", header = TRUE)
```

Instead of using the "final" (final.csv) dataset, we'll use a simpler file with source, destination, and weight data that has already been subsetted with filtered out locations. This was done using Excel. Though, in detail, to validate if nodeid contains the source's and destination's nodes and to remove un-used nodes, the Excel formula (=IF(ISERROR(VLOOKUP(A2,$F2:F$382, 1, FALSE)),FALSE,TRUE) and switching the F column to whichever column the source and destination is in, was used. It returns the values True or False for all sources and destinations. Using Excel's Filter function, the empty and false results were filtered out to clean the data.

Data Analysis

Data analysis is considered the calculations of centrality, betweenness and degree.

```
edges = read.csv("newChanneldata.csv", header = TRUE)
nodes = read.csv("nodesWithLoc.csv", header = TRUE)

colnames(edges)
colnames(edges)[1] <- "Source"
colnames(edges)[2] <- "Target"
colnames(edges)

colnames(nodes)
colToRemain <- c("NodeID", "alias")
nodes = nodes[colToRemain]

colnames(nodes)[1] <- "NodeID"
colnames(nodes)[2] <- "Alias"
colnames(nodes)

df1 <- sqldf("SELECT *
             FROM edges
             INNER JOIN nodes
             ON edges.Source = nodes.NodeID")
colnames(df1)
colnames(df1)[5] <- "SourceAlias"
colnames(df1)

colnames(edges)
colnames(nodes)

edges$weight <- edges[, 3] + 1
```

The nodes and edges is recrated to use the original dataframes. The weights of the edges need 1 added to each of the rows in order to eliminate all rows with a weightage of 0.

```
netGraph <- graph_from_data_frame(d = edges,
                                  vertices = nodes, directed = T)
netGraph
```

The statements above specify the dataframe to be used and create the graph object for data analysis. The graph object to be used for data analysis is netGraph

```
betweenness(netGraph, v = V(netGraph), directed = TRUE,
            weights = E(netGraph)$weight,
            nobigint = TRUE, normalized = FALSE)
```

This statement calculates the betweenness for the vertices. netGraph: graph to analyze v = V(netGraph): the vertices to calculate the vertex betweenness directed = TRUE: to consider the paths of the graph edges to be directed weights = E(netGraph)$weight: to calculate weighted betweenness nobigint = TRUE: to specify that there are many shortest paths between a pair of vertices normalized = FALSE: to normalize the betweenness scores

```
edge_betweenness(netGraph, e = E(netGraph), directed = TRUE,
                 weights =  E(netGraph)$weight)
```

Similar to the statement above but to calculate the edge_betweenness instead of the vertices' betweenness

```
closeness(netGraph, vids = V(netGraph), mode = ("total"),
          weights = E(netGraph)$weight, normalized = FALSE)
```

The closeness represents the steps required to get to another vertex from one specified vertex. netGraph: graph to analyze vids = V(netGraph): vertices to calculate the closeness mode = ("total"): define the paths to measure the distance in directed graphs, total uses all paths. weights = E(netGraph)$weight: to calculate weighted closeness normalized = FALSE: to calculate the normalized closeness

```
degree(netGraph, v = V(netGraph), mode = c( "all"),
       loops = FALSE, normalized = FALSE)
```

The degree represents the number of the adjacent edges of the nodes. netGraph: graph to analyze v = V(netGraph): nodes to calculate the degree mode = c("all"): all to represent the total of the in and out degree loops = FALSE: whether or not to count the loop edges, this does not need to be mentioned because the network has no loops normalized = FALSE: to calculate the normalized degree of nodes

```
transitivity(netGraph, type = ( "global"), vids = V(netGraph),
             weights = E(netGraph)$weight, isolates = c("NaN", "zero"))
```

Transitivity, also called the clustering coefficient, represents the probability that the adjacent nodes of a specified node is connected. netGraph: graph to analyze type = ("global"): the global transitivity, ratio of the triangles and the connected triples in the graph vids = V(netGraph): nodes to calculate the clustering coefficient weights = E(netGraph)$weight: to calculate the weighted transitivity isolates = c("NaN", "zero"): how to treat verticez with degree zero and one

Data Visualization

Please refer to the report for the respective images from the R code. Most of this code was obtained and modified from http://kateto.net/network-visualization. (Ognyanova, K. (2017) Network visualization with R. Retrieved from www.kateto.net/network-visualization)

With the data cleaned, the data visualization and analysis can now be done. Our project uses the igraph package in R. To use the igraph package in R, there are two specific files that are needed: nodes, edges. nodes: information on each node, especially the nodeid for each node with respective information edges: how each node ic connected, specified by the nodeid as connection: source, target, weight

Nodes is already specified as nodes. Edges is defined as the channels (newChanneldata)

```
edges <- channelsR
colnames(edges)[2] <- "target"
```

These two functions renames the dataframe 'channelsR' to 'edges' and changes the column of edges from 'destination' to 'target'.

```
net <- graph_from_data_frame(d = edges, vertices = nodes, directed = TRUE)
```

This function creates a graph object from the data frame specified previously. graph_from_data_frame: convert a data frame into a graph object. d = edges: signify the edges variable when creating the graph object vertices = nodes: signify the nodes variable directed = TRUE: there is a direction value from one node to another node so the newly created graph should be considered as a directed graph Finally, this newly created graph should be called "net"

```
net
```

From the result of running the above statement, the net graph has 381 nodes and 812 edges. From the node-edge attributes: vertex level character attributes: name, alias, ip, continent, country, region, city, Lat, Lng edge level numeric attribute: weight graph level character attribute: no columns in this graph data frame

```
plot(net)
```

This function simply plots the graph object. The problem with this graph is the vertex names. Too many vertex names are concentrated in one area so the nodes aren't visible.

```
plot(net, vertex.label = NA, vertex.size = 5, edge.color = "white", edge.lty
= 1, axes = TRUE)
```

In attempt to make the graph look more informative, the original plot statement is run with a few arguments. net: the graph object to plot vertex.label = NA: to not diplay the node label on the plot vertex.size = 5: size value of the vertex edge.color = "white": select the color of the edges as white to display the nodes are they are and not display the edges edge.lty = 1: select the line/edge type to be solid

```
plot(net, layout = layout_randomly, vertex.label = NA, vertex.size = 5,
edge.lty = 1, edge.color = "brown", edge.arrow.mode = 0)
```

This graph function displays the simple graph of the network showing the connections from one node to another node. net: the graph object to plot layout = layout_randomly: built-in layout to display the graph object randomly vertex.label = NA: to not displaiy the node label on the plot vertex.size = 5: size value of the vertex edge.lty = 1: select the line/edge type to be solid edge.color = "brown": display the edges to be the color brown edge.arrow.mode = 0: to not display the arrows of the edges

```
l <- layout_in_circle(net)
plot(net, layout = l, vertex.size = 1, edge.lty = 1, edge.color = "brown",
vertex.color = "blue", edge.arrow.mode = 0, vertex.label = NA)
```

This function runs another simple graph of the network showing the connections from one node to another node in a circular graph, hence called a "circle graph". l <- layout_in_circle(net): specify an object to create a circular layout of the graph object 'net' net: the graph object to plot. layout = l: specify the layout as the previously specified object layout vertex.size = 1: size value of the vertex edge.lty = 1: select the line/edge type to be solid edge.color = "brown": display the edges to be the color brown vertex.color = "blue": display the nodes to be the color blue edge.arrow.mode = 0: to not display the arrows of the edges vertex.label = NA: to not display the node label on the plot

```
l2 <- layout_on_sphere(net)
plot(net, layout = l2, vertex.size = 1, edge.lty = 1, edge.color = "brown",
vertex.color = "blue", edge.arrow.mode = 0, vertex.label = NA)
```

Based off the previous statement for the circle graph, the graph object is then plotted on a sphere graph.

```
l3 <- layout_with_fr(net)
plot(net, layout = l3, vertex.label = NA)
```

The function the net graph object with a force-directed network using the Fruchterman-Reingold algorithm. l3 <- layout_with_fr(net): specify an object to create a Fruchterman-Reingold graph of the graph object 'net' net: the graph object to plot vertex.label = NA: to not display the node label on the plot

```
tkid <- tkplot(net, vertex.label = NA)
plot(tkid)
```

This statement plots the net graph object with an GUI. It offers more expandability and options for the graphs and their data.

Data Visualization: simple graph using visNetwork package

```
nodes = read.csv("newChanneldata.csv", header = TRUE)
edges = read.csv("nodesWithLoc.csv", header = TRUE)

colnames(edges)[1] <- "Source"
colnames(edges)[2] <- "Target"
```

```
df1 <- sqldf("SELECT *
             FROM edges
             INNER JOIN nodes
             ON edges.Source = nodes.NodeID")

colnames(df1)[1] <- "Source"
colnames(df1)[5] <- "SourceAlias"


colnames(df1)[2] <- "nodeid"

colnames(df1)[4] <- "placeholder"
colnames(df1)[5] <- "SourceAlias"

df2 <- sqldf("SELECT *
             FROM df1
             INNER JOIN nodes
             on df1.nodeid = nodes.nodeid")

colnames(df2)[2] <- "Target"
colnames(df2)[7] <- "TargetAlias"

columnsRemain <- c("Source", "Target", "weight", "SourceAlias",
"TargetAlias")
df2 <- df2[columnsRemain]
```

In order to use the visNetwork package to plot graphs, a bit of data cleaning is required. These steps are the same steps required for the data analysis as they require the same data to be used for analysis and visualization.

```
g = graph.data.frame(df2, directed = TRUE)

visIgraph(g)
```

The graph data frame is created and specified as the object g and the visNetwork igraph graph is created.