Network Analysis

INFO 640 | Pratt Institute | McSweeney

Packages

```
#install.packages("igraph")
```

```
library(igraph)
```

We've made an edgelist listing the connections between a group of people. This is just a list of the 1:1 connections between each node If a node is connected to multiple other nodes, it will need an entry (row) for each connection.

```
f <- read.csv("Desktop/INFO640-Labs/Datasets/Sample_network.csv")
f
```

Right now it's just a dataframe. We need to tranform it into a matrix and turn it into a special data type: an edgelist. Now when we inspect it, the relationships are specified.

```
g <- graph.edgelist(as.matrix(f), directed=FALSE)
g
```

Now we can plot it to visualize the connections.

```
plot(g)
```

If we just want to know what the nodes (vertices) are, we can use the V function, to find the edges, use the E function.

```
V(g)
E(g)
```

Now let's say that we know some things about the entities in this network, maybe we know their age, their education level, or other demographic information. Let's first add their ages to each now. We will set vertex attributes and set edge attributes.

```
g <- set_vertex_attr(g, "age",
                     value = c(30, 45, 28, 33, 55, 42, 59))
vertex_attr(g)
```

Let's also say we know something about the strength of their relationships such as how many years they have worked together. Now can weight our edges based on how many years they have worked together.

```
g <- set_edge_attr(g, "years_working_together",
                   value = c(5, 6, 1, 1, 1, 10, 5, 3))
edge_attr(g)
```

It's more likely that you will have your data in a csv file than a list that you load in. These 2 files have the same data that we just manually entered (in a more comprehensible format) Now you can use these 2 files to make a graph from your data.

```
edges <- read.csv("../Datasets/edges.csv")
nodes <- read.csv("../Datasets/nodes.csv")

head(edges)
head(nodes)

my_graph <- graph_from_data_frame(d = edges, vertices = nodes, directed = FALSE)
plot(my_graph)
```

Now let's use formatting to visualize some of our data. We'll start with giving weights to the edges based on how many years people have worked together.

```
plot(my_graph, edge.width=E(my_graph)$years_working)
```

We can use this graph in a few ways. We can figure out who is the most connected person, what is the strongest connection. Now we will use our new graph to find everyone that Cece is connected to and everyone who has worked together for at least 5 years.

```
E(my_graph)[[inc('Cece')]]
E(my_graph)[[years_working>=5]]
```

It would be great if we could visualize some information about our employees. Let's visualize by their education level. First we will just encode who has a BA vs. an advanced degree. Here we just color every node who has a BA as blue and everyone else as green.

```
V(my_graph)$color <- ifelse(V(my_graph)$education == "BA", "blue","green")
plot(my_graph, vertex.label.color = "black")
```

To visualize 3 levels we will need to make a column for color and then create a vector of color. First transform the education column into a factor, then force those factors into numbers and then match the colors in your colors list to the 3 education levels.

Plot the results specifying the colors.

```
coul = c("red", "pink", "purple")
```

```
# Create a vector of color
my_color=coul[as.numeric(as.factor(V(my_graph)$education))]

plot(my_graph, vertex.color=my_color, vertex.label.color = "black")

#put it all together
plot(my_graph, edge.width=E(my_graph)$years_working, vertex.color=my_color, vertex.label.col
```

Now we'll use the size of the nodes to illustrate how connected a given node is. First we need to find out the degree of connectivity of our nodes. Then specify that connectivity as the size (this works because our nodes are ordered within our graph object).

```
degree(my_graph)
V(my_graph)$vertex_degree <- degree(my_graph)
plot(my_graph, vertex.size=V(my_graph)$vertex_degree)
plot(my_graph, vertex.size=V(my_graph)$vertex_degree*4) #the nodes were too small
plot(my_graph, edge.width=E(my_graph)$years_working, vertex.color=my_color, vertex.label.col
```

For more on the various parameters, available to you, visit: https://www.r-graph-gallery.com/248-igraph-plotting-parameters/

R offers a lot of different layouts, let's just illustrate one here:

```
plot(my_graph, layout = layout.fruchterman.reingold(my_graph))
```

Understanding the Connections

While visualizing networks is fun and quite beautiful, it's actually not always the most informative. Much of the power of networks comes from the ability to highlight to their users how nodes are connected in ways that may not be apparent. Obviously this is much more impactful with a larger dataset, but we will use this tpy one to illustrate our points.

We'll start by asking for all of Cece's neighbors. We did something similar above, but like many things with R, this method gives slightly different information. First, rather than asking for the edges to and from Cece (which then shows all the information about each one of those relationships), we've just asked for all of the names of the nodes connected to Cece. This is significant because it returns a list that you can then use in other functions (or plots) for other manipulations. For more on this, check out the ego_size documentation.

```
neighbors(my_graph, "Cece")
```

We can also find Cece's neighborhood. Maybe we want to know who she is connected to by just 2 steps away.

```
neighborhood(my_graph, order = 2, nodes = "Cece")
```

If you are thinking about how this type of insight drives essentially all social media, you would be right. This is the foundation of what LinkedIn sells - how many degrees away from your dream job are you? And how can they help you get there?!?

You can also find how 2 unconnected vertices are connected with the "intersection" function, but this is a little more complicated. Let's say we want to know how Cece and Erin are connected. We can see from our graph that they are only one degree removed from each other. But let's assume that we are building a networking site and don't necessarily know that. We want to find all of Cece's neighbors and all of Erin's neighbors and see where they overlap - find their mutual connections.

```
c <- neighbors(my_graph, "Cece")
e <- neighbors(my_graph, "Erin")
```

```
intersection(c,e)
```

Let's assume instead that we want to see who the least connected people are, that is, we want to see the longest path. This will give us a measurement of how disconnected or spread out our network is.

```
farthest_vertices(my_graph)
```

It looks like Bo and Fran are the least connected - 3 connections away from each other. You may notice that Don and Gus are also 3 lengths removed. The farthest vertices functions only returns the first furthest length, so the meaningful bit here is the 3, not the nodes.

Another useful measurement is the ego of each node. The ego is how many people can be reached in n steps.

```
ego(my_graph, 2, "Cece")
```

For example, everyone in the graph can be reached within 2 steps of Cece: Cece is 0 edges away, Abby, Bo, Erin and Gus are 1 edge away, and Don & Fran are 2.

Centrality

There are many ways to calculate centrality. The most straightforward may be the degree - or the closeness measure. This measures which node has the most direct connections. Another interesting metric is betweenness, which measures how often a given node appears on the shortest paths within the network. This measures how important a node is as a bridge between networks. The normalized betweenness gives a relative weight for your particular network.

```
degree(my_graph)
betweenness(my_graph, normalized = FALSE)
```

Though this is a very small network, we can already see a difference in the betweenness and the raw number of connections a node has. Abby, Erin, and Cece all have the same number of connections, but Cece and Erin's betweenness is much high since they are both far more important to the overall connections of the network. The are more central to the network. Fran, Gus, Don, and Bo are inessential to the flow of information: they don't connect to anyone but the central group of nodes, and therefore their betweenness is 0.0.

To find the most linearly connected vector, calculate the degree centrality and sort the values.

```
deg_g <- degree(my_graph)
deg_g_sort <- sort(deg_g, decreasing = TRUE)
deg_g_sort[1:5] #print only the first 5
```

To calculate who has the greatest sphere of influence, we can use the eigenvector centrality. The eigenvector centrality accounts for how interconnected the connections are. "Where the centrality of each actor is proportional to the sum of the centralities of those actors to whom he or she is connected." We'll use the same technique here of sorting by the largest value.

```
eig_g <- eigen_centrality(my_graph)
eig_g_sort <- sort(eig_g[[1]], decreasing=T)
eig_g_sort[1:5]
```

Significance

To determine if a metric is significantly different from the others, run an experiment with n graphs, and compare where the metric of the target graph falls. For example, if we thought that the mean distance in our target network was particularly long, we may want to compare it to what the mean distance would be in the network were randomly ordered. The best way to do this is to create very many simulated networks. We will run 1000 simulations of our target network.

This is about to get vvery boring with our current network. There is a bitcoin trust network on the LMS. It is very slow to load and will take a long time. Feel free to continue with the toy network if you are working on a tablet or very slow device. For simplicity's sake, I am going to call the new graph my_graph1.

SNAP https://snap.stanford.edu/data/soc-sign-bitcoin-alpha.html

```
bitcoin <- read.csv("../Datasets/soc-sign-bitcoinalpha.csv")
head(bitcoin)
length(bitcoin)

my_graph1 <- graph_from_data_frame(d = bitcoin, directed = FALSE)
plot(my_graph1, vertex.size=.01, layout=layout.auto, vertex.label=NA)
```

First, we'll calculate some basic metrics about our network as a whole,

```
edge_density(my_graph1)
mean_distance(my_graph1)
```

Given that this is a bitcoin trust, we expect that the distance between individuals is further than it would be by chance. Let's check.

First make a list of number 1:1000 to iterate through. Then set up a for loop and on every trip through those numbers, call a random reorganization (with the erdos.renyi.game function) and create the network. Store that network in a list of networks.

```
l <- vector('list', 1000)
for (i in 1:1000){
  l[[i]] <- erdos.renyi.game(
    n = gorder(my_graph1),
    p.or.m = edge_density(my_graph1),
    type="gnp")
}
```

Then we can use the networks stored in this vector of networks to get a distribution of the mean_distances for a network of this size. We can plot this on a histogram and then see where our network ends up (you may be surprised)

```
l.apls <- unlist(lapply(l, mean_distance, directed=FALSE))

hist(l.apls, breaks=20)
abline(v=mean_distance(my_graph1, directed = FALSE), col='red', lty=3)
```

Our network is so distinct from what would occur by chance that the v line doesn't even appear on the histogram. It is statistically significant.

There is a lot more we could do with this network, such as finding the ego of the most central node, or the diameter of the network.