Part1_Q1

April 6, 2020

Part 1 - Generating Random Networks

Question 1 - Create random networks using Erdos-Renyi (ER) model

[17]:
```
# # Install Necessary Packages
# install.packages("igraph")
# install.packages("Matrix")
# install.packages("pracma")

# Load Packages
library('igraph')
library('Matrix')
library('pracma')
```

a) Create undirected random networks with n = 1000 nodes, and the probability p for drawing an edge between two arbitrary vertices 0.003, 0.004, 0.01, 0.05, and 0.1. Plot the degree distributions. What distribution is observed? Explain why. Also, report the mean and variance of the degree distributions and compare them to the theoretical values.

[19]:
```
p_array = c(0.003,0.004,0.01,0.05,0.1)
n = 1000
for (p in p_array)
{
    g <- erdos.renyi.game(n, p, type="gnp", directed=F);
    png(sprintf("plots/part1/question1/q1a_dist_p_%5.3f.png",p))
    plot(degree.distribution(g), main=sprintf("Degree distribution of the␣
↪network with p=%5.3f",p),ylab="Probability",xlab="Degree")
    lines(degree.distribution(g))
    dev.off()
    png(sprintf("plots/part1/question1/q1a_hist_p_%5.3f.png",p))
    hist(degree(g), col=rgb(0.1,0.5,1,.6), xlab="Degree",ylab="Frequency",␣
↪main=paste("Histogram for p=",p))
    dev.off()
    print(sprintf("p=%5.3f: mean=%5.3f, Variance=%5.3f", p, mean(degree(g)),␣
↪var(degree(g))))
}
```

```
[1] "p=0.003: mean=3.002, Variance=3.167"
```

```
[1] "p=0.004: mean=4.156, Variance=4.370"
[1] "p=0.010: mean=10.162, Variance=9.964"
[1] "p=0.050: mean=49.750, Variance=50.574"
[1] "p=0.100: mean=99.766, Variance=92.786"
```

b) For each p and n = 1000, answer the following questions: Are all random realizations of the ER network connected? Numerically estimate the probability that a generated network is connected. For one instance of the networks with that p, find the giant connected component (GCC) if not connected. What is the diameter of the GCC?

[17]:
```
for (p in p_array){
    count <- 0
    for (i in seq(1,1000,1)){
        g <- erdos.renyi.game(n, p, type="gnp", directed = F)
        if (is.connected(g)){
            count <- count + 1
        }
    }
    print(sprintf("Results for p = %5.3f",p))
    print(sprintf("Probability that the network is connected is: %5.3f",count/
↪1000))

    # GCC
    g.components <- clusters(g)

    # Get the largest component
    ix <- which.max(g.components$csize)

    # Get the subgraph corresponding to the giant component
    g.giant <- induced.subgraph(g, which(g.components$membership == ix))

    print(sprintf("Is the graph connected: %s", is.connected(g)))
    print(sprintf("Number of nodes of the GCC: %d", vcount(g.giant)))
    print(sprintf("Number of Edges of the GCC: %d", ecount(g.giant)))
    print(sprintf("Diameter of the GCC: %5.3f",diameter(g.giant)))
    cat("\n")
}
```

```
[1] "Results for p = 0.003"
[1] "Probability that the network is connected is: 0.000"
[1] "Is the graph connected: FALSE"
[1] "Number of nodes of the GCC: 932"
[1] "Number of Edges of the GCC: 1496"
[1] "Diameter of the GCC: 17.000"

[1] "Results for p = 0.004"
[1] "Probability that the network is connected is: 0.000"
[1] "Is the graph connected: FALSE"
```

```
[1] "Number of nodes of the GCC: 982"
[1] "Number of Edges of the GCC: 2046"
[1] "Diameter of the GCC: 11.000"

[1] "Results for p = 0.010"
[1] "Probability that the network is connected is: 0.955"
[1] "Is the graph connected: TRUE"
[1] "Number of nodes of the GCC: 1000"
[1] "Number of Edges of the GCC: 4955"
[1] "Diameter of the GCC: 6.000"

[1] "Results for p = 0.050"
[1] "Probability that the network is connected is: 1.000"
[1] "Is the graph connected: TRUE"
[1] "Number of nodes of the GCC: 1000"
[1] "Number of Edges of the GCC: 24737"
[1] "Diameter of the GCC: 3.000"

[1] "Results for p = 0.100"
[1] "Probability that the network is connected is: 1.000"
[1] "Is the graph connected: TRUE"
[1] "Number of nodes of the GCC: 1000"
[1] "Number of Edges of the GCC: 50010"
[1] "Diameter of the GCC: 3.000"
```

c) It turns out that the normalized GCC size (i.e., the size of the GCC as a fraction of the total network size) is a highly nonlinear function of p, with interesting propertiesvoccurring for values where p = O(1/n) and p = O((ln n)/n). For n = 1000, sweep over values of p from 0 to a pmax that makes the network almost surely connected and create 100 random networks for each p. pmax should be roughly determined by yourself. Then scatter plot the normalized GCC sizes vs p. Plot a line of the average normalized GCC sizes for each p along with the scatter plot.

i. Empirically estimate the value of p where a giant connected component starts to emerge (define your criterion of "emergence")? Do they match with theoretical values mentioned or derived in lectures?

ii. Empirically estimate the value of p where the giant connected component takes up over 99% of the nodes in almost every experiment.

[19]:
```
#1c: Normalized GCC
# Scatter plot normalized GCC for 100 random networks for each p.

p_min = 0
p_max = 0.01
p_step = 0.0001
p_array = seq(p_min,p_max,p_step)
n = 1000
num_networks = 100
```

```
i = 0

rand_networks = seq(1,num_networks)
len_gcc = length(p_array)*length(rand_networks)
gcc.sizes = vector(mode="numeric", length=len_gcc)
p_plot = rep(p_array, times=1, each=num_networks) # Create 100 networks for␣
 ↪each p


for (p in p_array){
    # generate 100 random netwoks
    for (x in rand_networks ){
        i = i+1
        g = erdos.renyi.game(n, p, type="gnp")
        # GCC
        g.components = clusters(g)
        # Get the largest component
        ix = which.max(g.components$csize)
        # Get the subgraph corresponding to the giant component
        g.giant = induced.subgraph(g, which(g.components$membership == ix))
        gcc.sizes[i] = length(V(g.giant))

    }
}
png(sprintf("plots/part1/question1/q1c_normalizedGCC.png"), width = 1000,␣
 ↪height = 600)
# options(repr.plot.width=10, repr.plot.height=5)
plot(p_plot,gcc.sizes/n,main="Normalized GCC size vs.␣
 ↪p",xlab="p",ylab="normalized GCC size")
axis(side=1, at=seq(0,0.01,by=0.001))
lines(gcc.sizes)
# box()
dev.off()
```

png: 2

[20]:
```
#1c: Mean Normalized GCC
p_min = 0
p_max = 0.02
p_step = 0.0001
p_array = seq(p_min,p_max,p_step)
n = 1000
num_networks = 100
i = 0
len_gcc = length(p_array)

rand_networks = seq(1,num_networks)
```

```
gcc.sizes = vector(mode="numeric", length=len_gcc)

for (p in p_array){
    i = i+1
    # generate 100 random netwoks
    gcc.size.sum = 0
    for (x in rand_networks ){

        g = erdos.renyi.game(n, p, type="gnp")
        # GCC
        g.components = clusters(g)
        # Get the largest component
        ix = which.max(g.components$csize)
        # Get the subgraph corresponding to the giant component
        g.giant = induced.subgraph(g, which(g.components$membership == ix))

        gcc.size.sum = gcc.size.sum + length(V(g.giant))
    }

    # find mean value of GCC Size.
    gcc.sizes[i] = gcc.size.sum/num_networks

}

png(sprintf("plots/part1/question1/q1c_AverageNormalizedGCC.png"), width =␣
 ↪1000, height = 600)
plot(p_array,gcc.sizes/n,main="Average normalized GCC size vs.␣
 ↪p",xlab="p",ylab="Average normalized GCC size", type="l")
dev.off()
```

png: 2

[21]:
```
print(sprintf("Theoretical p Value at which GCC emerges: %5.3f",1/1000))
print(sprintf("Theoretical p Value at which GCC takes over 99%% of nodes:␣
 ↪%8f",log(1000, base = exp(1))/1000))
```

```
[1] "Theoretical p Value at which GCC emerges: 0.001"
[1] "Theoretical p Value at which GCC takes over 99% of nodes: 0.006908"
```

d) i.Define the average degree of nodes c = n × p = 0.5. Sweep over the number of nodes, n, ranging from 100 to 10000. Plot the expected size of the GCC of ER networks with n nodes and edge-formation probabilities p = c/n, as a function of n. What trend is observed?

ii. Repeat the same for c = 1.

iii. Repeat the same for values of c = 1.1,1.2,1.3, and show the results for these three values in a single plot.

iv. What is the relation between the expected GCC size and n in each case?

[30]:
```
#1d: Degree of nodes - c
#i) c  = 0.5

c = 0.5
n_start = 100
n_end = 10000
n_step = 100
range_n = seq(n_start,n_end,n_step)
gcc.sizes = vector(mode="numeric", length=length(range_n))
i = 0
num_networks = 50
rand_networks = seq(1,num_networks)

for (n in range_n){
    i = i+1
    p = c/n
    gcc.size.sum = 0

    # average over 50 random networks
    for( a in rand_networks){
        g = erdos.renyi.game(n, p, type="gnp")
        # GCC
        g.components = clusters(g)
        ## Get the largest component
        ix = which.max(g.components$csize)
        # Get the subgraph corresponding to the giant component
        g.giant = induced.subgraph(g, which(g.components$membership == ix))

        gcc.size.sum = gcc.size.sum+vcount(g.giant)

    }

    # add to vector
    gcc.sizes[i] = (gcc.size.sum/num_networks)

}

png(sprintf("plots/part1/question1/q1d_expectedGCC_c_%5.3f.png",c))
plot(range_n,gcc.sizes,main= paste("Expected Size of GCC vs. n for c␣
 ↪=",c),xlab="n",ylab="Expected Size of GCC")
lines(lowess(range_n,gcc.sizes), col="red")
dev.off()
```

png: 2

[31]:
```
#1d: Degree of nodes - c
#ii) c  = 1
```

```
c = 1
n_start = 100
n_end = 10000
n_step = 100
range_n = seq(n_start,n_end,n_step)
gcc.sizes = vector(mode="numeric", length=length(range_n))
i = 0
num_networks = 50
rand_networks = seq(1,num_networks)

for (n in range_n){
    i = i+1
    p = c/n
    gcc.size.sum = 0

    # average over 50 random networks
    for( a in rand_networks){
        g = erdos.renyi.game(n, p, type="gnp")
        # GCC
        g.components = clusters(g)
        ## Get the largest component
        ix = which.max(g.components$csize)
        # Get the subgraph corresponding to the giant component
        g.giant = induced.subgraph(g, which(g.components$membership == ix))

        gcc.size.sum = gcc.size.sum+vcount(g.giant)

    }

    # add to vector
    gcc.sizes[i] = (gcc.size.sum/num_networks)

}
png(sprintf("plots/part1/question1/q1d_expectedGCC_c_%5.3f.png",c))
plot(range_n,gcc.sizes,main= paste("Expected Size of GCC vs. n for c␣
↪=",c),xlab="n",ylab="Expected Size of GCC")
lines(lowess(range_n,gcc.sizes), col="red")
dev.off()
```

png: 2

[32]:
```
#1d: Degree of nodes - c
#iii) c  = 1.1,1.2,1.3

n_start = 100
n_end = 10000
```

```
n_step = 100
range_n = seq(n_start,n_end,n_step)
gcc.sizes = vector(mode="numeric", length=length(range_n))
num_networks = 50
rand_networks = seq(1,num_networks)

c = 1.1
i = 0
for (n in range_n){
    i = i+1
    p = c/n
    gcc.size.sum = 0

    # average over 50 random networks
    for( a in rand_networks){
        g = erdos.renyi.game(n, p, type="gnp")
        # GCC
        g.components = clusters(g)
        ## Get the largest component
        ix = which.max(g.components$csize)
        # Get the subgraph corresponding to the giant component
        g.giant = induced.subgraph(g, which(g.components$membership == ix))

        gcc.size.sum = gcc.size.sum+vcount(g.giant)

    }

    # add to vector
    gcc.sizes[i] = (gcc.size.sum/num_networks)

}
png(sprintf("plots/part1/question1/q1d_expectedGCC_c_%5.3f.png",c))
plot(range_n,gcc.sizes,col='red',main= "Expected Size of GCC vs. n (for␣
 ↪different Degree of nodes c)",xlab="n",ylab="Expected Size of GCC")


c = 1.2
i = 0
for (n in range_n){
    i = i+1
    p = c/n
    gcc.size.sum = 0

    # average over 50 random networks
    for( a in rand_networks){
        g = erdos.renyi.game(n, p, type="gnp")
        # GCC
        g.components = clusters(g)
        ## Get the largest component
        ix = which.max(g.components$csize)
        # Get the subgraph corresponding to the giant component
        g.giant = induced.subgraph(g, which(g.components$membership == ix))

        gcc.size.sum = gcc.size.sum+vcount(g.giant)

    }

    # add to vector
    gcc.sizes[i] = (gcc.size.sum/num_networks)

}

points(range_n,gcc.sizes,col='blue')


c = 1.3
i = 0
for (n in range_n){
    i = i+1
    p = c/n
    gcc.size.sum = 0

    # average over 50 random networks
    for( a in rand_networks){
        g = erdos.renyi.game(n, p, type="gnp")
        # GCC
        g.components = clusters(g)
        ## Get the largest component
        ix = which.max(g.components$csize)
        # Get the subgraph corresponding to the giant component
        g.giant = induced.subgraph(g, which(g.components$membership == ix))

        gcc.size.sum = gcc.size.sum+vcount(g.giant)

    }

    # add to vector
    gcc.sizes[i] = (gcc.size.sum/num_networks)

}

points(range_n,gcc.sizes,col='black')
legend(100, 1000, legend=c("c = 1.1", "c = 1.2","c = 1.3"),
       col=c("red", "blue","black"), lty=1:2, cex=0.8)
```

```
dev.off()
```

png: 2