(CSC 495) Homework 5

Kai Chung, Ying

Spring 2017

Some visualizations of the Dining data

```r
library(knitr)
setwd('/Users/KevQuant/Desktop/Depaul/csc495/wk5/hw5')

read_chunk("hwk5.R")
knitr::opts_chunk$set(echo = TRUE)
```

Part I: Simplifying the network

Step 1: Load the necessary libraries and data

```r
# Load packages
library("ggplot2")
library("GGally")
library("mcclust")
# Must load other packages first
library("sand")
```

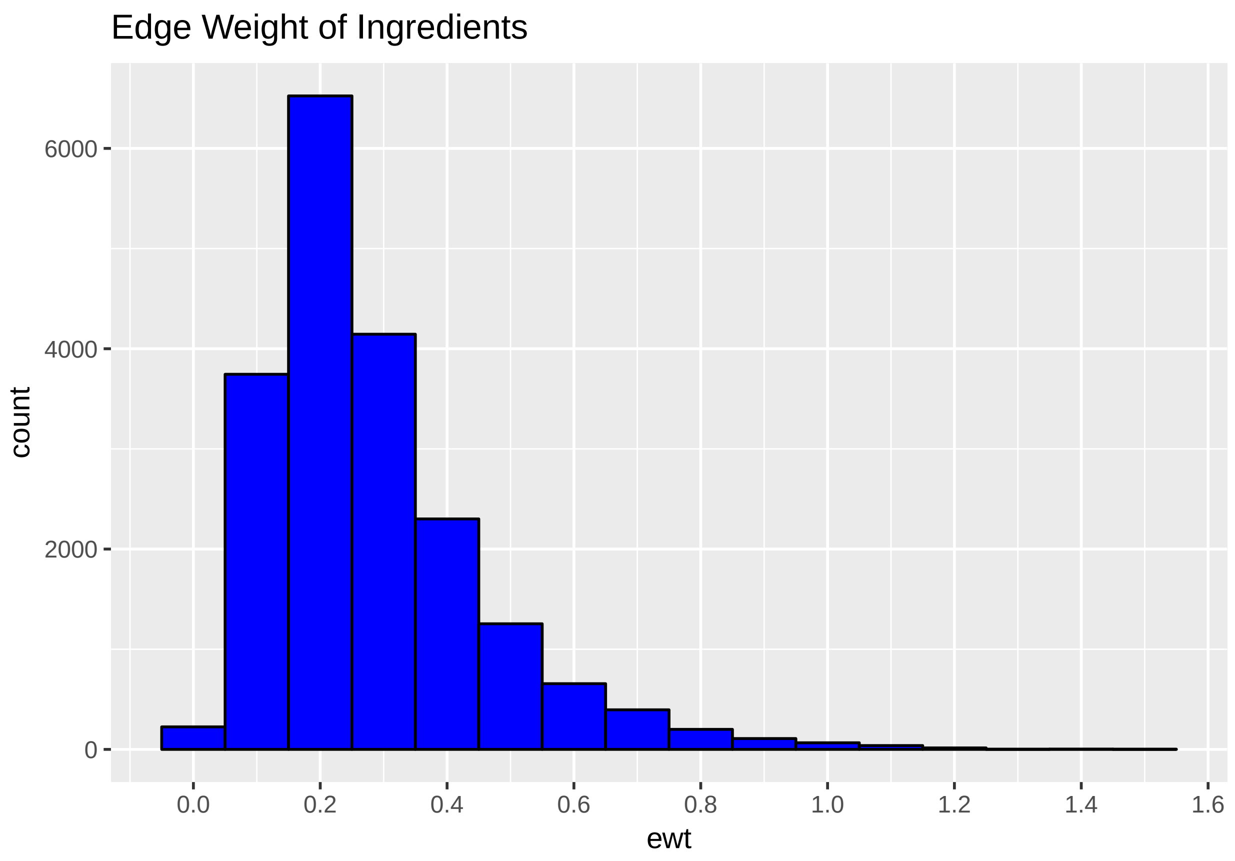
Step 2: Load the data and summarize

```r
setwd("/Users/KevQuant/Desktop/Depaul/csc495/wk5/hw5")
ingred <- read.graph("ingred.graphml", format="graphml")
summary(ingred)
```

```
## IGRAPH U-W- 1106 19679 --
## + attr: label (v/c), id (v/c), weight (e/n)
```

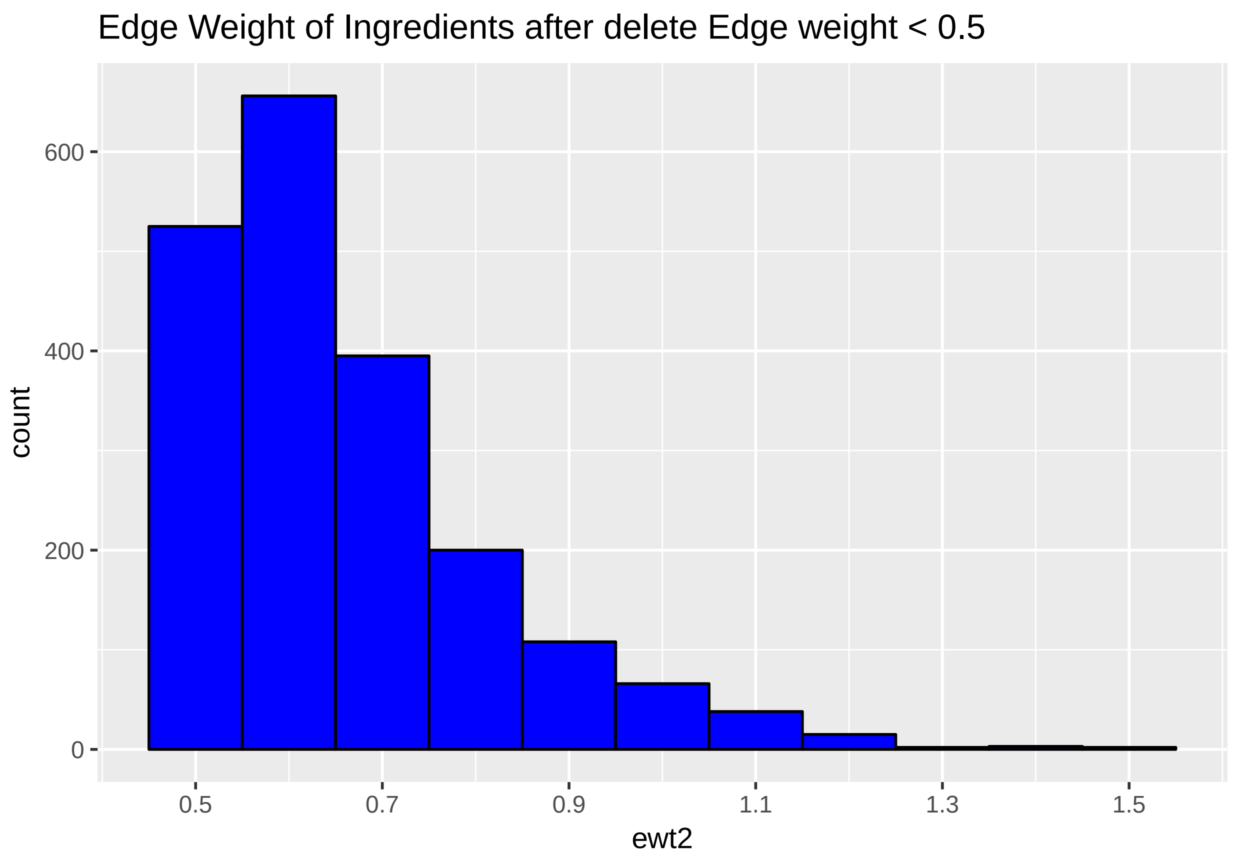
Step 3: Create a histogram of edge weight.

```r
g1<-ggplot(data.frame(ewt=E(ingred)$weight),aes(x=ewt))
g1<-g1+geom_histogram(binwidth = 0.1, col="black",fill="blue" )
g1<-g1+ggtitle('Edge Weight of Ingredients')
g1<-g1+scale_x_continuous(breaks=seq(0,2,0.2))
print(g1)
```



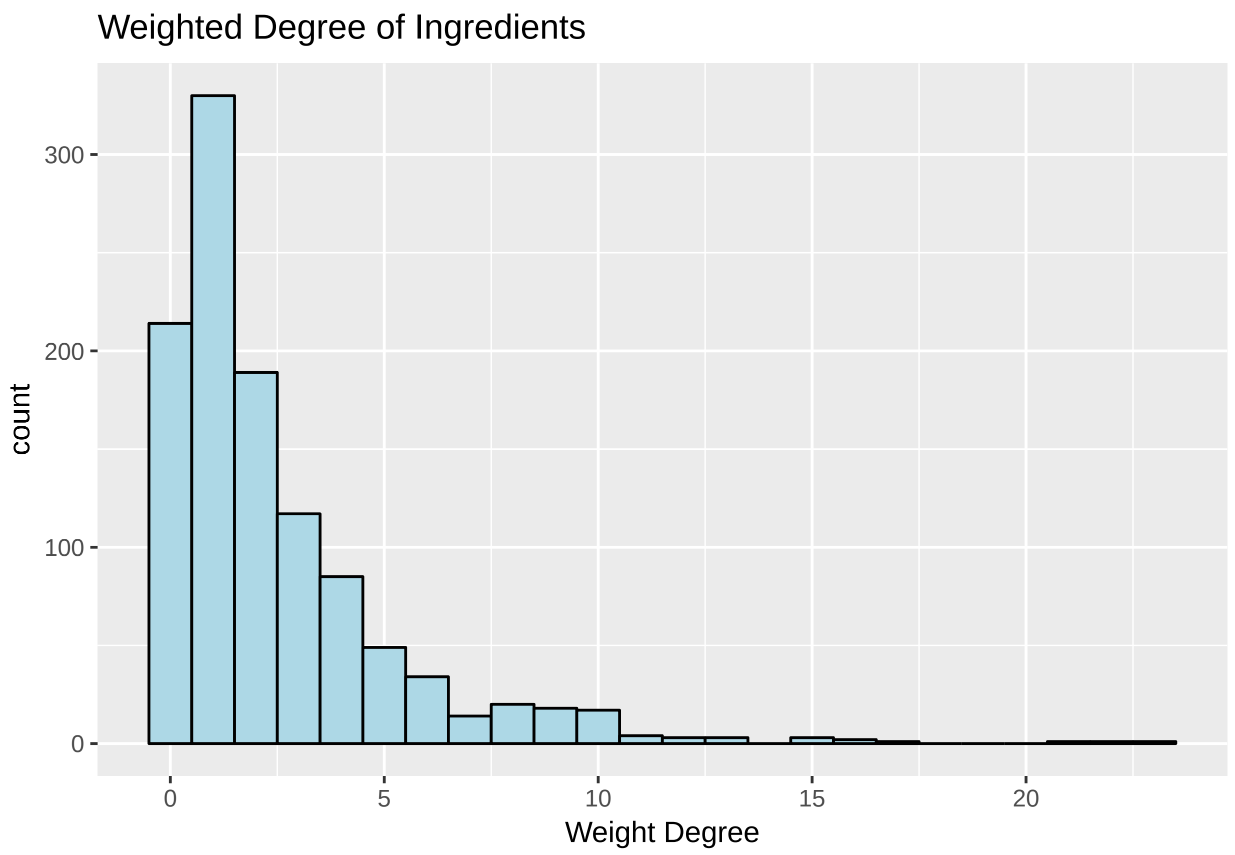
Step 4: Remove the edges with weight less than 0.50.

```
#Remove the Edges with weight less than 0.5
ingred2<-delete.edges(ingred,E(ingred)[E(ingred)$weight<0.5])
#Check range of the Edge Weight of ingred2, the minimum value should be 0.5
g2<-ggplot(data.frame(ewt2=E(ingred2)$weight),aes(x=ewt2))
g2<-g2+geom_histogram(binwidth = 0.1, col="black",fill="blue" )
g2<-g2+ggtitle('Edge Weight of Ingredients after delete Edge weight < 0.5')
g2<-g2+scale_x_continuous(breaks=seq(0.5,2,0.2))
print(g2)
```



Step 5: Create a histogram of weighted degree (graph.strength)

```
ingred2_wt_deg<-graph.strength(ingred2)
g3<-ggplot(data.frame(wt_deg=ingred2_wt_deg), aes(x=wt_deg))
g3<-g3+geom_histogram(binwidth = 1,col="black",fill="lightblue")
g3<-g3+ggtitle("Weighted Degree of Ingredients")
g3<-g3+xlab('Weight Degree')
g3<-g3+scale_x_continuous(breaks = seq(0,30,5))
print(g3)
```



Step 6: Remove the vertices with weighted degrees less than or equal to 2.0

```
ingred3<-delete_vertices(ingred2,V(ingred2)[ingred2_wt_deg<=2.0])
```

Step 7: Remove singleton nodes.

```
ingred3<-delete_vertices(ingred3,V(ingred3)[degree(ingred3)==0])
#Recheck the summary of ingred3 network
summary(ingred3)
```

```
## IGRAPH U-W- 440 1313 --
## + attr: label (v/c), id (v/c), weight (e/n)
```

Part II: Community detection

Step 8: Set the random seed to a fixed value, such as your birthday. This will ensure that the random aspects of the clustering will work the same way each time.

```
set.seed(20170428)
```

Step 9: Run five community detection algorithms on the network (use weights)

1. Leading eigenvector
2. Fastgreedy
3. Edge betweenness (this will be slow)
4. Walktrap (default # of steps = 4)
5. Walktrap (steps = 10)

```
le<-cluster_leading_eigen(ingred3, weights = E(ingred3)$weight)
fg<-cluster_fast_greedy(ingred3,weights = E(ingred3)$weight)
bt<-cluster_edge_betweenness(ingred3,weights = E(ingred3)$weight)
wt4<-cluster_walktrap(ingred3,steps=4, weights = E(ingred3)$weight)
wt10<-cluster_walktrap(ingred3,steps=10,weights = E(ingred3)$weight)

#Compare the different clusterings based on length (the number of clusters found) and modularity.
as.numeric(lapply(list(le,fg,bt,wt4,wt10),length))
```

```
## [1]  9 15 18 23 27
```

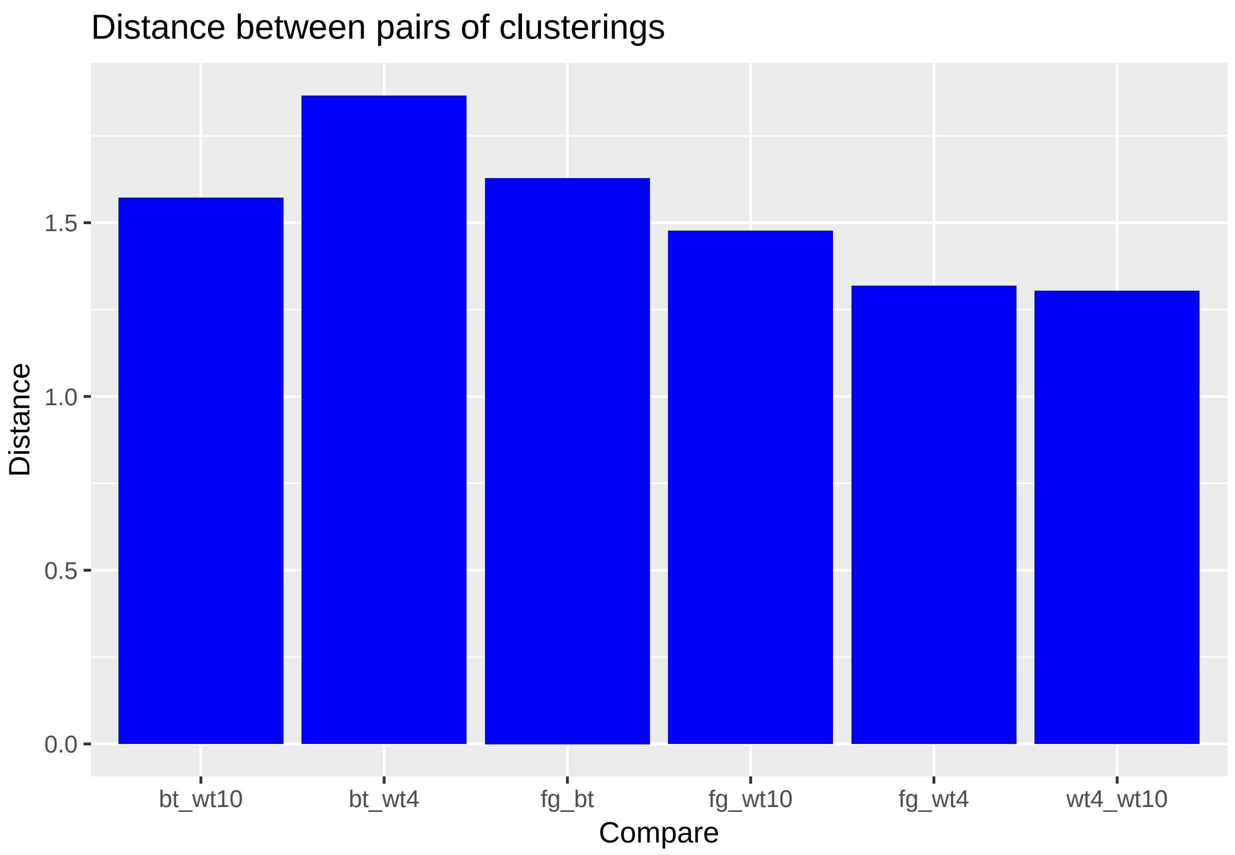
```
as.numeric(lapply(list(le,fg,bt,wt4,wt10),modularity))
```

```
## [1] 0.6531344 0.7520854 0.7470404 0.7237128 0.7420002
```

```
#Based on the above,the lowest modularity is from cluster_leading_eigen
```

Step 10: With the remaining four metrics (Drop out the lowest Modularity), use the vi.dist() method from the mcclust package to compute the variation of information distance between all pairs of clusterings (6 pairs total). Plot as a bar plot.

```
Dist<-c(vi.dist(membership(fg),membership(bt)),
        vi.dist(membership(fg),membership(wt4)),
        vi.dist(membership(fg),membership(wt10)),
        vi.dist(membership(bt),membership(wt4)),
        vi.dist(membership(bt),membership(wt10)),
        vi.dist(membership(wt4),membership(wt10)))
Dist_df<-data.frame(Compare=c("fg_bt",'fg_wt4','fg_wt10','bt_wt4'
                              ,'bt_wt10','wt4_wt10'),Dist=Dist)
g4<-ggplot(Dist_df)
g4<-g4+geom_bar(mapping = aes(x = Compare, y = Dist),
                stat = "identity",fill="blue")
g4<-g4+ggtitle("Distance between pairs of clusterings")
g4<-g4+ylab('Distance')
print(g4)
```



```
#Based on the above plot, the value walktrap4 & walktrap10 is the closest to ZERO which
#means these 2 clusters is the most similar to each other among other comparison.
#On the other hand, the value of edge_betweenness and walktrap 4 is the largest, so that
#the difference between them is the largest.
```

Step 11: Question: Fast Greedy has the highest modularity, which is not surprising since it directly optimizes that. Identify the algorithm with the next highest modularity and use that for the next part of the assignment.

```
as.numeric(lapply(list(le,fg,bt,wt4,wt10),modularity))
```

```
## [1] 0.6531344 0.7520854 0.7470404 0.7237128 0.7420002
```

The next highest modularity is Edge-Betweenness

Part III: Visualization

Step 12: Use which to locate an ingredient that you're familiar with.

```
## Pick 'Cereal' as my ingredient
which(V(ingred3)$label=='cereal')
```

```
## [1] 314
```

Step 13: Extract the clusters containing your ingredient from the fastgreedy and your other best algorithm into separate networks using induced.subgraph.

```r
## 'Cereal' is in cluster #6 from the fast greedy algorithm
communities(fg)
```

```
## $`1`
##  [1] 136 137 139 141 142 144 233 243 244 281 298 303 320 342 344 395 396
## 
## $`2`
##  [1]  39  40  42  66  75  93 152 199 209 211 217 220 222 232 234 236 238
## [18] 240 249 270 274 275 287 294 327 330 426
## 
## $`3`
##  [1]  28  30  47  72  90 100 153 173 174 175 176 178 185 226 304 305 306
## [18] 308 309 310 315 316 318 321 359 360 361 362 363 364 365 373 400 403
## [35] 425 436
## 
## $`4`
##  [1]  32  67  79 128 133 134 143 145 177 179 182 184 186 187 189 192 193
## [18] 194 195 196 197 200 203 204 205 218 224 229 264 268 279 307 340 341
## [35] 345 384 408 412 413 415 429 437
## 
## $`5`
##  [1]  31  60  62  69  74  84  86 118 146 147 159 160 161 162 163 164 166
## [18] 167 168 169 180 181 198 201 202 210 219 223 225 235 239 241 253 254
## [35] 255 260 265 266 267 273 280 285 301 302 328 366 367 377 406 419 430
## 
## $`6`
##  [1]  10  11  14  16  18  20  22  27  35  37  43  55  85  89  92  94  97
## [18] 102 107 109 110 111 114 117 119 120 121 122 207 283 312 313 314 325
## [35] 355 356 378 397 401 402 404 407 410
## 
## $`7`
##  [1]  58  59  61  63  64  65  68  70  73  76  77  78  80  81  82  83 132
## [18] 215 227 251 252 256 258 261 262 271 282 284 288 289 290 291 343
## 
## $`8`
##  [1]  71 130 170 190 213 230 317 335 353 354 428 431
## 
## $`9`
##  [1]   9  12  17  23  26  33  36  38  41  44  46  52  53  54  56 106 124
## [18] 135 140 151 191 206 245 276 277 278 286 292 300 329 336 348 352 357
## [35] 368 369 370 371 372 374 375 376 383 385 387 388 389 390 391 392 393
## [52] 394 399 422 438 439 440
## 
## $`10`
##  [1] 131 138 148 149 150 263 297 311 319 322 386 409 424
## 
## $`11`
##  [1]  48 129 171 172 183 188 208 212 214 216 221 228 242 295 296 326 379
## [18] 411 414 421
## 
## $`12`
##  [1]   1   2   3   4   5   6   7  15  21  24  25  29  34  45  49  50  51
## [18]  57 108 154 246 247 248 250 257 259 269 299 323 324 331 332 333 334
## [35] 337 338 339 346 347 349 350 351 358 380 381 382 398 405 418 423 427
## [52] 432 433 434 435
##
## $`13`
##  [1]  96 103 104 105 112 115 116 123 125 126 127 155 156 157 158 272 416
## [18] 417
##
## $`14`
##  [1]   8  13  19  87  91  95  98  99 101 113 165 231 237 420
##
## $`15`
## [1]  88 293
```

```r
communities(fg)[[6]]
```

```
##  [1]  10  11  14  16  18  20  22  27  35  37  43  55  85  89  92  94  97
## [18] 102 107 109 110 111 114 117 119 120 121 122 207 283 312 313 314 325
## [35] 355 356 378 397 401 402 404 407 410
```

```r
## 'Cereal' is in cluster #5 from the edge-betweenness algorithm
communities(bt)
```

```
## $`1`
##  [1]   1   3   5   6   7  24  34  51 108 151 246 248 257 299 323 324 331
## [18] 332 333 334 337 346 347 349 350 351 380 423 432 433 434 435
##
## $`2`
##  [1]   2   4  15  22  25  29  43  45  49  50  57  92  93  94  97 114 154
## [18] 247 250 259 269 270 338 339 358 378 381 382 398 401 405 418
##
## $`3`
##  [1]   8  13  19  91  95  98  99 101 113 165 420
##
## $`4`
##  [1]   9  12  21  26  36  41  44  53  56 106 124 135 357 422 427 439
##
## $`5`
##  [1]  10  11  14  16  18  20  27  33  35  37  38  46  47  52  55  85  89
## [18] 102 107 109 110 111 117 119 120 121 122 207 283 312 313 314 355 356
## [35] 397 399 402 404 407 410
##
## $`6`
##  [1]  17  23  54 140 167 191 198 201 202 206 227 245 276 277 278 286 292
## [18] 300 301 302 329 336 348 352 366 367 368 369 370 371 372 374 375 376
## [35] 383 385 387 388 389 390 391 392 393 394 406 419 438 440
##
## $`7`
##  [1]  28 100 173 174 175 176 318 400 403 425
##
## $`8`
##  [1]  30  71  72  90 145 153 226 304 305 306 308 309 310 315 316 317 321
## [18] 359 360 361 362 363 364 365 373 428 436
##
## $`9`
##  [1]  31  60  62  69  86  87  88 118 146 147 159 160 161 162 163 164 166
## [18] 168 169 180 181 219 223 225 231 235 237 239 253 254 255 260 265 266
## [35] 267 273 280 293 328 377
##
## $`10`
##  [1]  32  67  79 128 133 134 148 177 178 179 182 184 185 186 187 189 192
## [18] 193 194 195 196 197 200 203 204 205 218 224 229 241 264 268 279 307
## [35] 340 341 345 384 408 412 413 415 429 437
##
## $`11`
## [1]  39 220 222 232 236 325
##
## $`12`
##  [1]  40  66  75 152 199 209 211 215 217 234 238 240 249 274 275 287 294
## [18] 327 330 426
##
## $`13`
## [1]  42 190 213 230 353 354
##
## $`14`
##  [1]  48 129 171 172 183 188 208 212 214 216 221 228 242 296 326 379 411
## [18] 414 421
##
## $`15`
##  [1]  58  61  63  64  65  68  70  73  74  76  77  78  80  81  82  83  84
## [18] 130 132 170 251 252 256 258 261 262 271 282 284 285 288 290 291 335
## [35] 343 431
##
## $`16`
##  [1]  59  96 103 104 105 112 115 116 123 125 126 127 155 156 157 158 210
## [18] 272 416 417 430
##
## $`17`
##  [1] 131 136 137 138 141 142 143 144 149 150 233 244 263 281 289 295 297
## [18] 298 303 311 319 320 322 386 409 424
##
## $`18`
## [1] 139 243 342 344 395 396
```

```r
communities(bt)[[5]]
```

```
##  [1]  10  11  14  16  18  20  27  33  35  37  38  46  47  52  55  85  89
## [18] 102 107 109 110 111 117 119 120 121 122 207 283 312 313 314 355 356
## [35] 397 399 402 404 407 410
```

```r
##Extract the 2 communities includes 'Cereal' node with id=314
fg_sub<-induced_subgraph(ingred3,communities(fg)[[6]])
bt_sub<-induced_subgraph(ingred3,communities(bt)[[5]])
```

Step 14: Plot each of the networks in igraph.

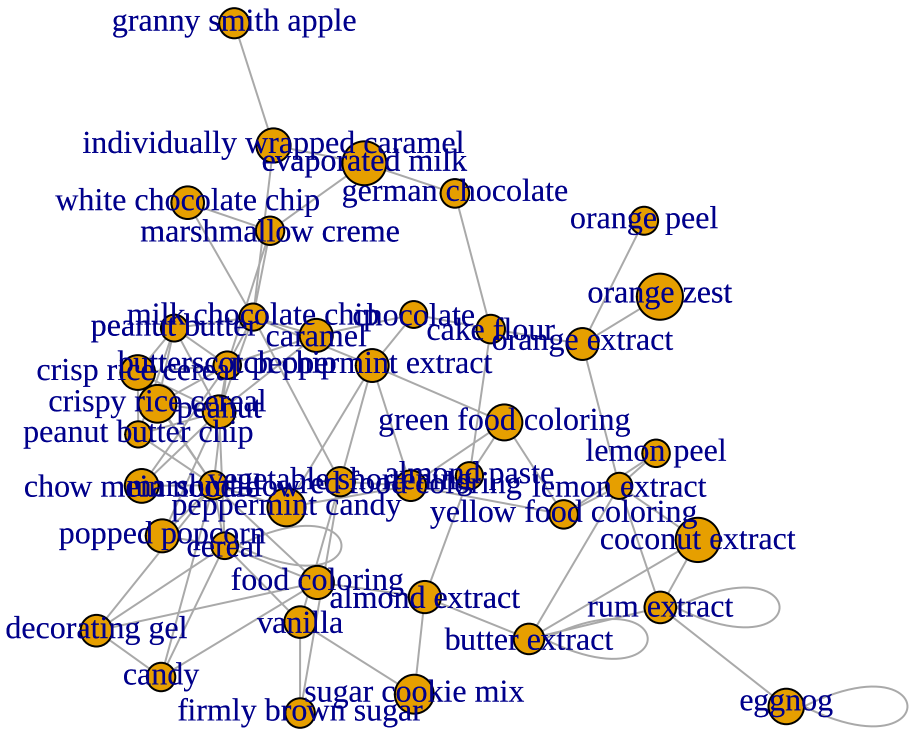
```r
par(mfrow=c(1,1))
par(mar=c(0.5,0.5,0.5,0.5))
plot(fg_sub,
     layout=layout_with_kk,
     vertex.label.cex=1,
     vertex.size=E(fg_sub)$weight*15)
```

```
## Warning in vertex.size + 6 * 8 * log10(nchar(labels) + 1): longer object
## length is not a multiple of shorter object length

## Warning in layout[, 1] + label.dist * cos(-label.degree) * (vertex.size + :
## longer object length is not a multiple of shorter object length

## Warning in vertex.size + 6 * 8 * log10(nchar(labels) + 1): longer object
## length is not a multiple of shorter object length

## Warning in layout[, 2] + label.dist * sin(-label.degree) * (vertex.size + :
## longer object length is not a multiple of shorter object length
```



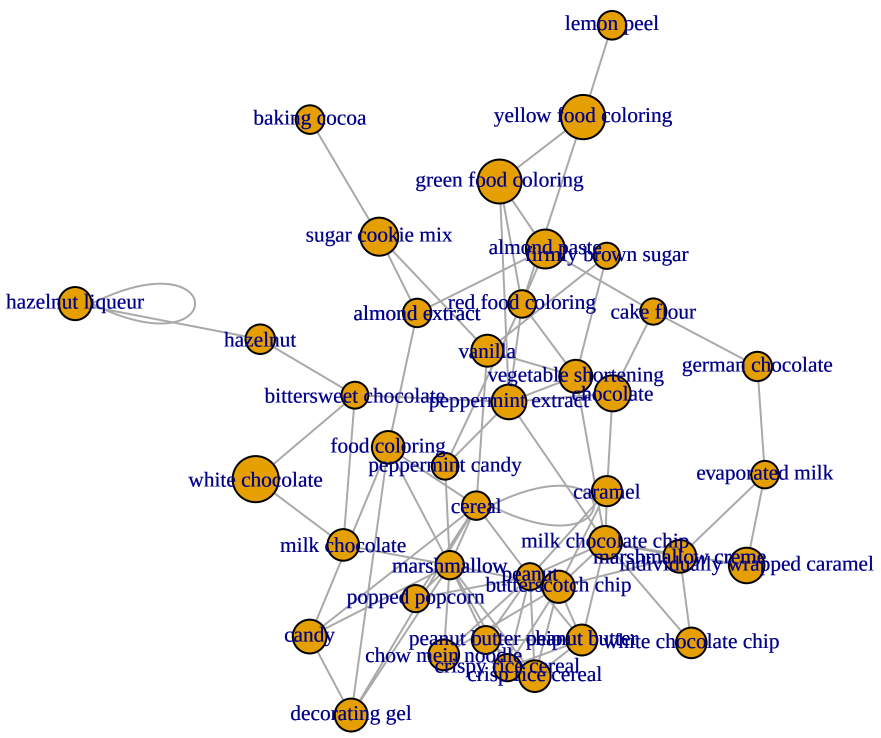
```r
plot(bt_sub,
     layout=layout_with_kk,
     vertex.label.cex=0.7,
     vertex.size=E(bt_sub)$weight*15)
```

```
## Warning in vertex.size + 6 * 8 * log10(nchar(labels) + 1): longer object
## length is not a multiple of shorter object length

## Warning in layout[, 1] + label.dist * cos(-label.degree) * (vertex.size + :
## longer object length is not a multiple of shorter object length

## Warning in vertex.size + 6 * 8 * log10(nchar(labels) + 1): longer object
## length is not a multiple of shorter object length

## Warning in layout[, 2] + label.dist * sin(-label.degree) * (vertex.size + :
## longer object length is not a multiple of shorter object length
```

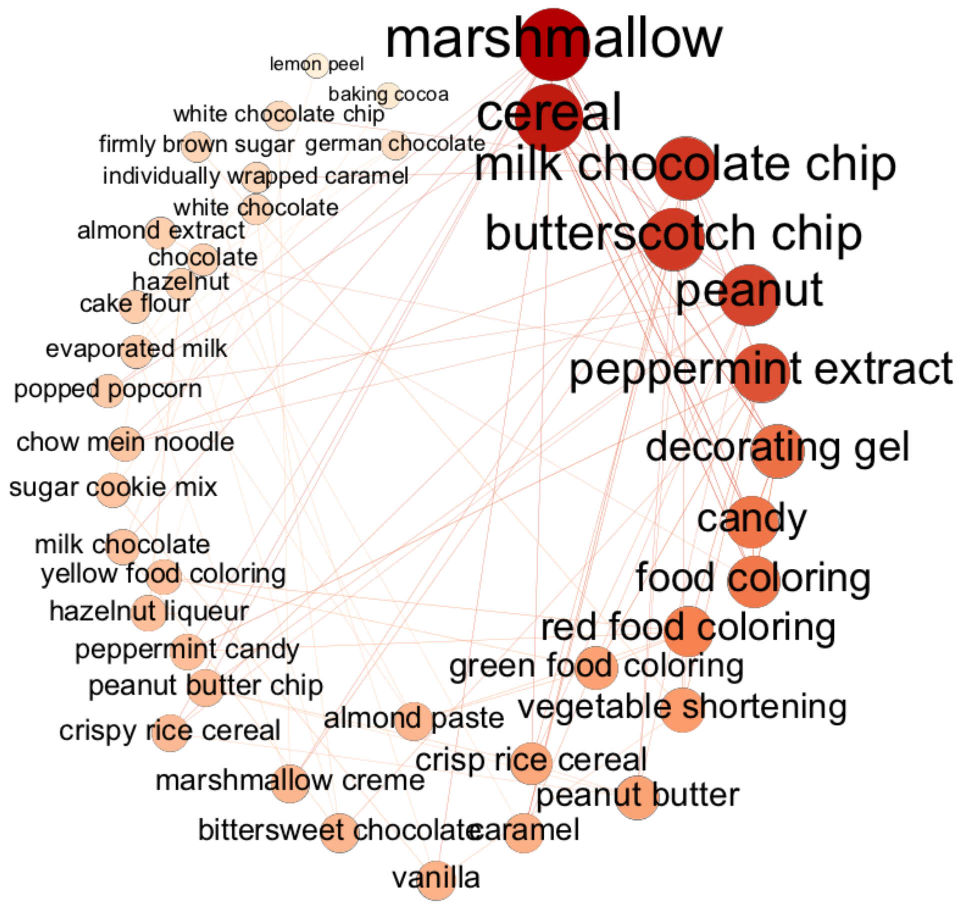


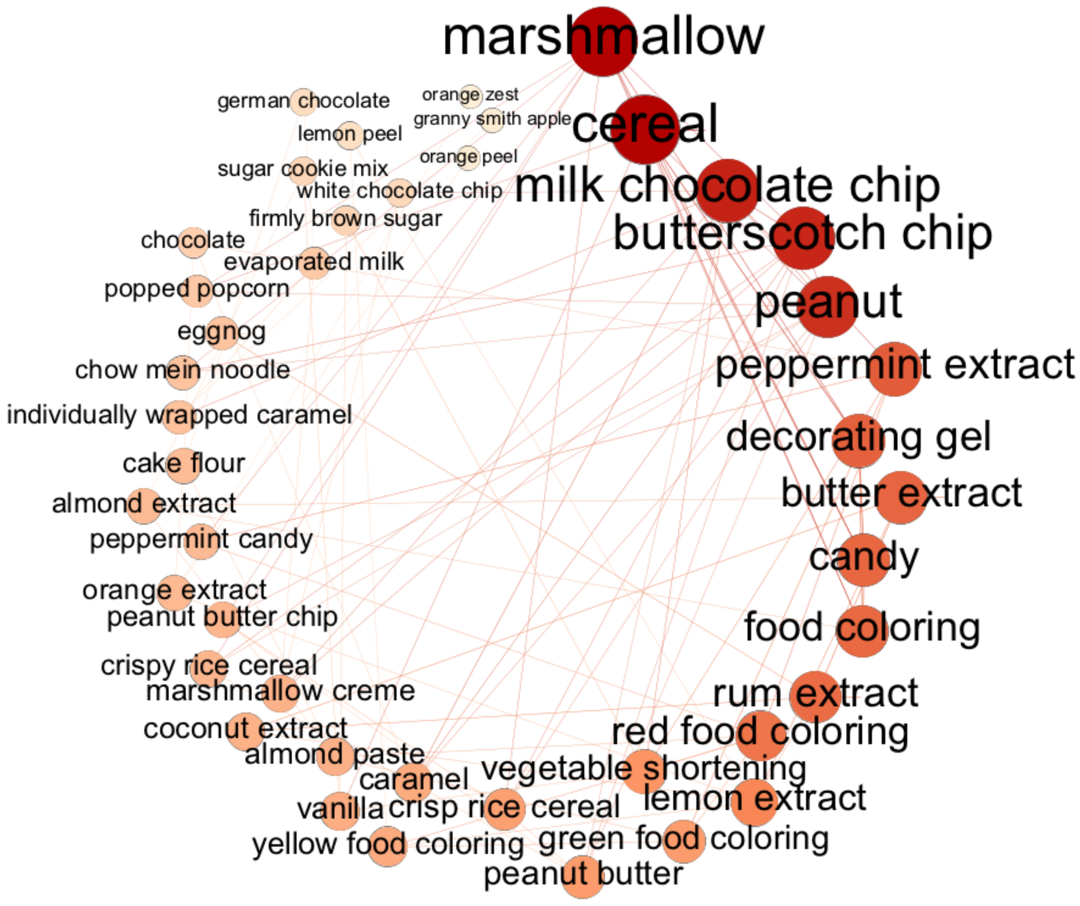
Step 15: Export these networks to graph files and plot in Gephi, including the labels and sizing the nodes by weighted degree.

```
write_graph(fg_sub,"fg_sub.graphml",format="graphml")
write_graph(bt_sub,"bt_sub.graphml",format="graphml")
```

Step 16: Save the images in two PNG files: food1.png and food2.png.

Step 17: Embed the food1.png and food2.png files in your R markdown using the image markdown syntax:





Step 18: Question: What type of food(s) do the clusters represent?

Based on the above plots, I would say that both clusters are representing topping ingredients. Their elements among the communities are very similar. For my sense, these ingredients would usually add to the top of the final food products, e.g. decorating gel, food coloring, marshmallow creme, caramell, beanut butter etc. Also, most of them are Sweet too. I would imagine they could make Smoothies or Ice-cream type of food.

Step 19: Question: Which cluster captures the associated cuisine better? Why? (They could be equally good.)

I would say that both clusters are representing similar ingredients except small portions of them are snack type of food (e.g.chow mein noodle and peanut butter chip etc). I think that all those clusters are quite accurate because their clusters' elements are very much serve as similar Topping type of ingredients.

If I need to choose a better cuisine out of these 2 clusters. I would choose the fast greedy cluster because I personally like more ingredients to add to the food because it would let me taste more favor in different ways, especially sweet Dessert. These 2 cluster which make me imagine ice-cream because those sweet and coloring ingredients are perfect match to ice-cream.

Also, we could observe that the fast greedy provides higher value of modularity which means includes more elements in the cluster.

For my choice of picking cereal, we could see that Cereal proportionally has the higher weighted degree comparing with other elements. This means that Cereal might be a very popular ingredients for food infusion, e. And I believe that a lot people would agree to that. So that overall I agree with the Community Detection Algorithm based on this result.