A First Look at the Triplicates & Model Fitting

Filipe Russo

March 24, 2019

Introduction

Our aim is to prepare the Proteomics dataset of the microalgae Chrorella vulgaris in such a way that we can construct a network of co-expressed proteins. For this objective we chose to adapt the `R` package `WGCNA` (Weighted Gene Co-expression Network Analysis), we need it to work with proteins instead of genes.

Loading & Cleaning

First, we load our original dataset `proteinGroups.txt`, store it in the variable `Proteins` and do some previous seen cleanup:

```r
library(dplyr) # for data manupilation
library(stringr) # for string manipulation
library(WGCNA) # load the WGCNA package
library(tidyr) # for spread function (long to wide)

# The following setting is important for WGCNA, do not omit.
options(stringsAsFactors = FALSE)

# little nice function to check if an element IS NOT in a list
'%ni%' <- Negate('%in%')

# 1.a Loading Expression Data

Proteins = read.csv("proteinGroups.txt", sep = "\t", header=TRUE)

names(Proteins) <- str_replace(names(Proteins),
                               pattern = "MixoTP4",
                               replacement = "MixoTP6")
names(Proteins) <- str_replace(names(Proteins),
                               pattern = "HeteTP4",
                               replacement = "HeteTP10")

# we define positive identifiers for the Reverse,
# Only.identified.by.site
# and Potential.contaminant columns
rev_posids = list("+")
site_posids = list("+")
cont_posids = list("+")

clean_proteins <- Proteins %>%

  # filters out rows based on posids of rev, site and cont
  # filters out rows based on the protein IDs, names with CON or REV
  filter(Reverse %ni% rev_posids &
         Only.identified.by.site %ni% site_posids &
```

```
        Potential.contaminant %ni% cont_posids &
        !str_detect(Majority.protein.IDs, "CON|REV")) %>%

  # keeps only the ids and the LFQ columns
  select(c("Majority.protein.IDs", str_subset(names(Proteins), "LFQ"))) %>%
  rename(ID = Majority.protein.IDs)
```

Hierarchical Clustering

Through *Hierarchical Clustering* we will be able to cluster our triplicates and visualize sample outliers among them.

```
# removes the ID column, transposes the clean_proteins dataframe and
# then it turns the matrix back to a data.frame class object
datExprA = as.data.frame(t(clean_proteins[, -c(1:1)]))

# picks the protein ids from clean_proteins ID column and
# stores it as column names in the datExprA data.frame
names(datExprA) = clean_proteins$ID

# picks the triplicate names stored in the clean_proteins columns and
# saves them as row names from datExprA
rownames(datExprA) = names(clean_proteins)[-c(1:1)]

# 1.b Checking data for excessive missing values and
# Identification of outlier microarray samples

gsg = goodSamplesGenes(datExprA, verbose = 3);
```

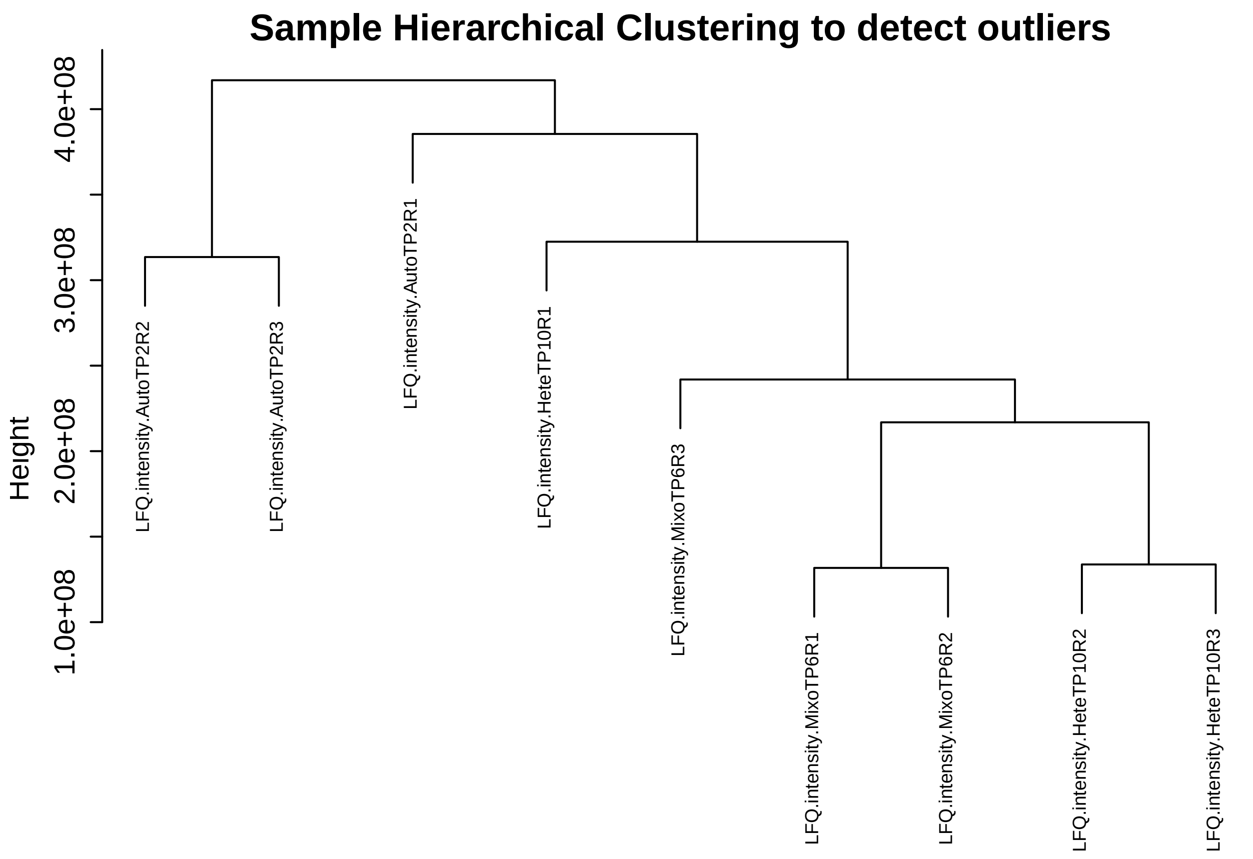
```
##  Flagging genes and samples with too many missing values...
##   ..step 1
```

```
gsg$allOK
```

```
## [1] TRUE
```

```
if (!gsg$allOK)
{
  # Optionally, print the gene and sample names that were removed:
  if (sum(!gsg$goodGenes)>0)
    printFlush(paste("Removing proteins:", paste(names(datExprA)[!gsg$goodGenes],
                                                 collapse = ", ")));
  if (sum(!gsg$goodSamples)>0)
    printFlush(paste("Removing samples:", paste(rownames(datExprA)[!gsg$goodSamples],
                                                collapse = ", ")));

  # Remove the offending genes and samples from the data:
  datExprA = datExprA[gsg$goodSamples, gsg$goodGenes]
}

# Clustering dendrogram of samples based on their Euclidean distance
sampleTree1 = hclust(dist(datExprA), method = "average");

# pdf(file = "sampleClustering.pdf", width = 12, height = 9);
```

```r
par(cex = 0.6);
par(mar = c(0,4,2,0))
plot(sampleTree1,
     main = "Sample Hierarchical Clustering to detect outliers",
     sub = "",
     xlab = "",
     cex.lab = 1.5,
     cex.axis = 1.5,
     cex.main = 2)
```



It seems the sample HeteTP10R1 is an outlier. We suppose this happened due to the zeros found in some LFQ intensities, some represent true zeros and others represent missing values. We want the samples to be clustered in 3 different clusters: Auto, Mixo, Hete; lets see how the `cutree` function handles it.

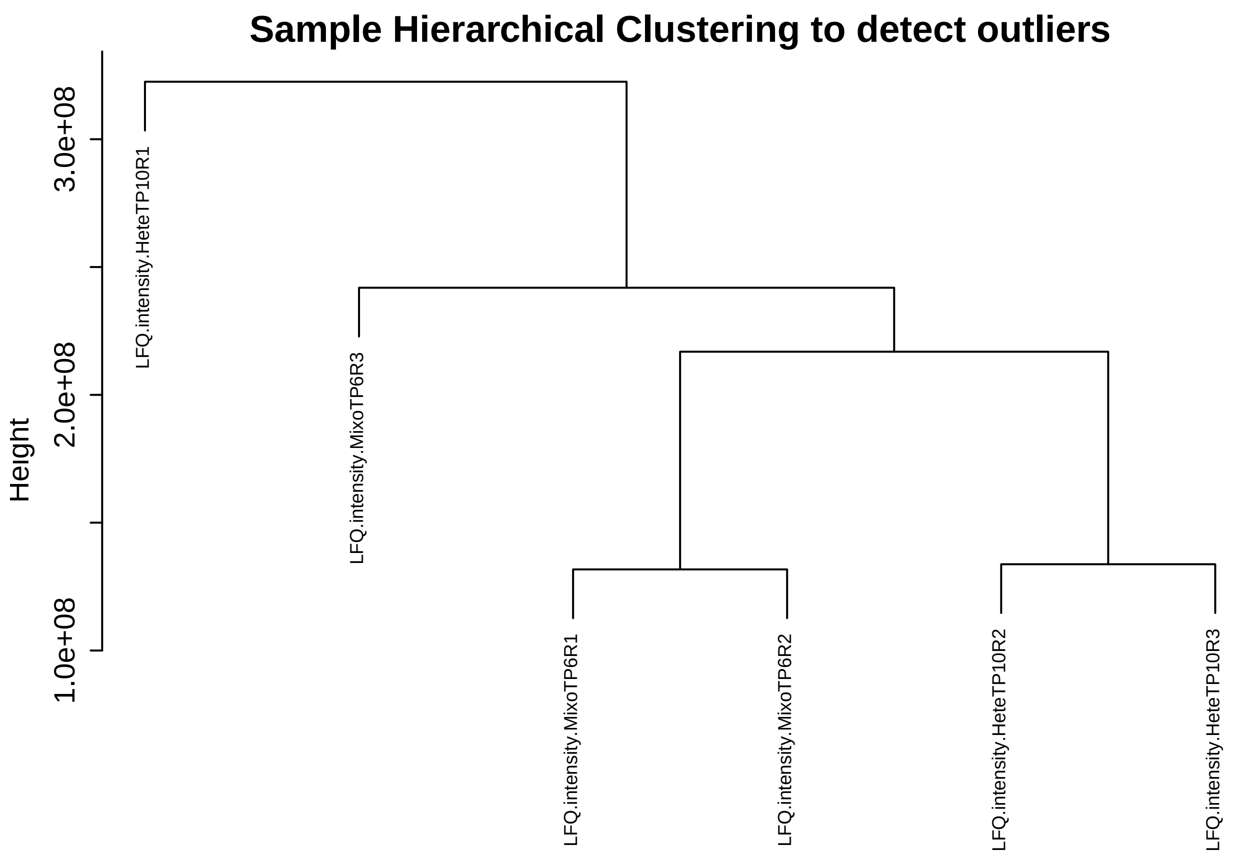
```r
cutree(sampleTree1, k = 3)
```

```
##  LFQ.intensity.AutoTP2R1  LFQ.intensity.AutoTP2R2  LFQ.intensity.AutoTP2R3
##                        1                        2                        2
## LFQ.intensity.HeteTP10R1 LFQ.intensity.HeteTP10R2 LFQ.intensity.HeteTP10R3
##                        3                        3                        3
##  LFQ.intensity.MixoTP6R1  LFQ.intensity.MixoTP6R2  LFQ.intensity.MixoTP6R3
##                        3                        3                        3
```

The result suggests the sample AutoTP2R1 is an outlier among the AutoTP2 replicates and the growth conditions Heterotrophic and Mixotrophic seem to be more similar to each other than to the Autotrophic one.

Lets remove the Autotrophic growth condition from the data.frame and see how `hclust` and `cutree` handle the samples:

```r
# Clustering dendrogram of samples based on their Euclidean distance
sampleTree2 = hclust(dist(datExprA[-c(1:3), ]), method = "average");

# pdf(file = "sampleClustering.pdf", width = 12, height = 9);
par(cex = 0.6);
par(mar = c(0,4,2,0))
plot(sampleTree2,
     main = "Sample Hierarchical Clustering to detect outliers",
     sub = "",
     xlab = "",
     cex.lab = 1.5,
     cex.axis = 1.5,
     cex.main = 2)
```

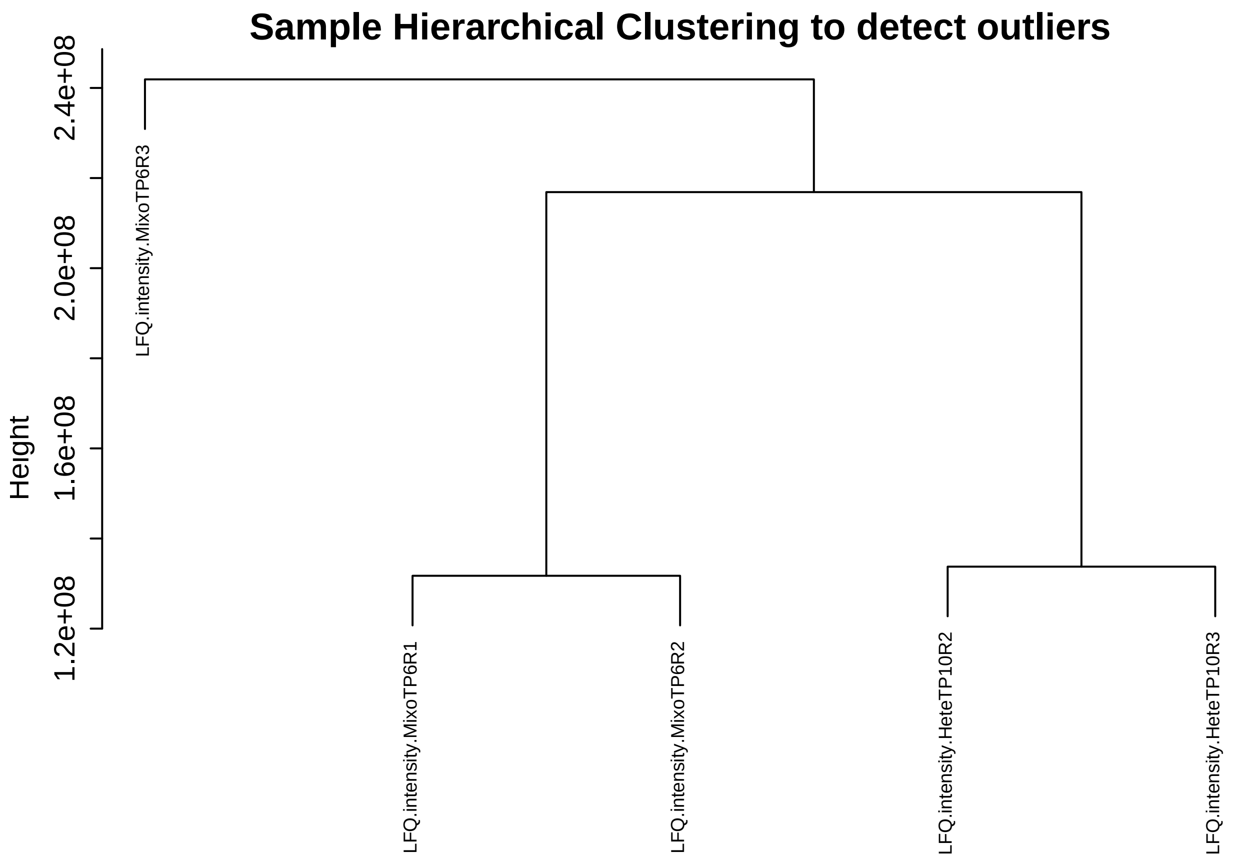


```r
cutree(sampleTree2, k = 2)
```

```
## LFQ.intensity.HeteTP10R1 LFQ.intensity.HeteTP10R2 LFQ.intensity.HeteTP10R3
##                         1                         2                         2
##  LFQ.intensity.MixoTP6R1  LFQ.intensity.MixoTP6R2  LFQ.intensity.MixoTP6R3
##                         2                         2                         2
```

As expected the HeteTP10R1 sample seems to be an outlier, lets remove it and redo the process:

```r
# Clustering dendrogram of samples based on their Euclidean distance
sampleTree3 = hclust(dist(datExprA[-c(1:4), ]), method = "average");

# pdf(file = "sampleClustering.pdf", width = 12, height = 9);
par(cex = 0.6);
par(mar = c(0,4,2,0))
plot(sampleTree3,
     main = "Sample Hierarchical Clustering to detect outliers",
     sub = "",
     xlab = "",
     cex.lab = 1.5,
     cex.axis = 1.5,
     cex.main = 2)
```

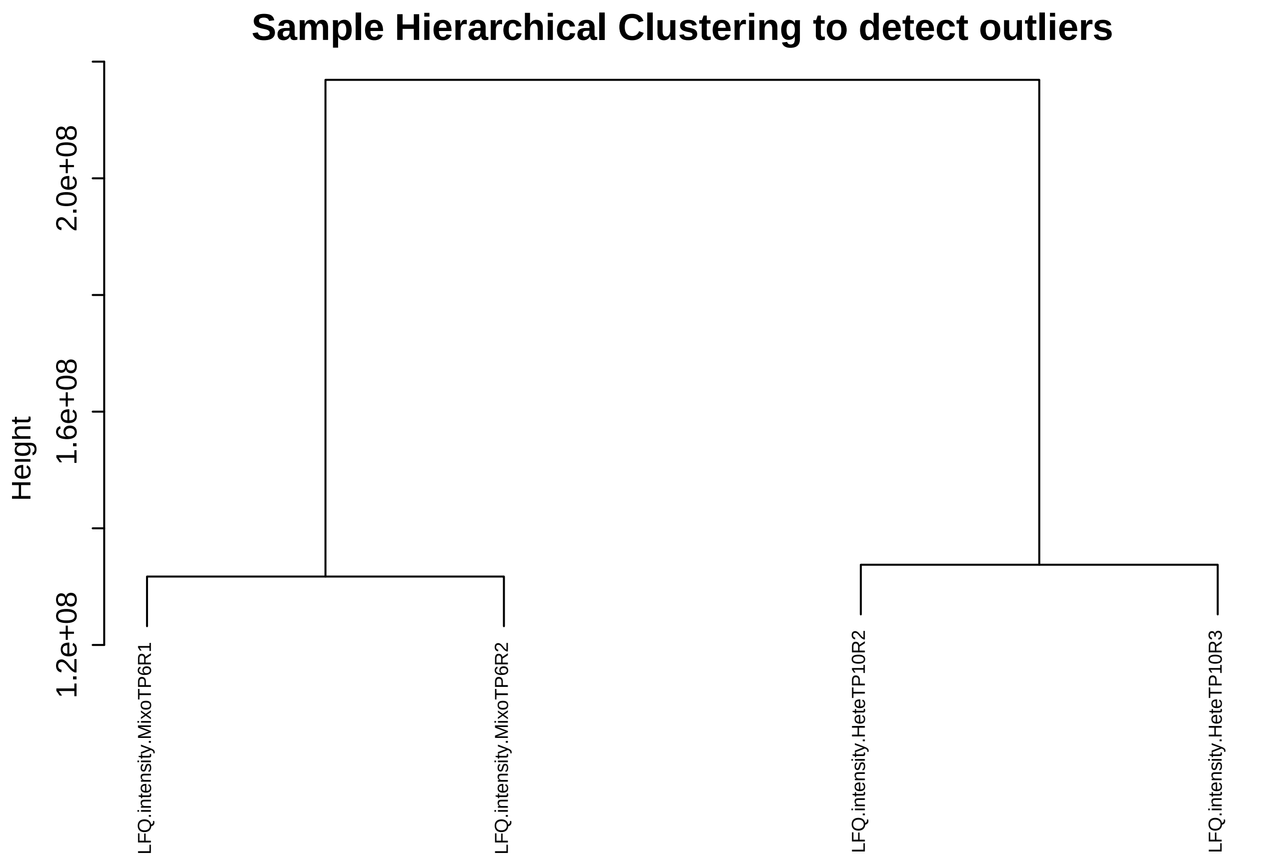


```r
cutree(sampleTree3, k = 2)
```

```
## LFQ.intensity.HeteTP10R2 LFQ.intensity.HeteTP10R3  LFQ.intensity.MixoTP6R1
##                        1                        1                        1
##  LFQ.intensity.MixoTP6R2  LFQ.intensity.MixoTP6R3
##                        1                        2
```

Now, the sample MixoTP6R3 seems to be the outlier. Let's remove it, take one last look at this slicing process and move on with the analysis:

```r
# Clustering dendrogram of samples based on their Euclidean distance
sampleTree4 = hclust(dist(datExprA[-c(1:4, 9), ]), method = "average");

# pdf(file = "sampleClustering.pdf", width = 12, height = 9);
par(cex = 0.6);
par(mar = c(0,4,2,0))
plot(sampleTree4,
     main = "Sample Hierarchical Clustering to detect outliers",
     sub = "",
     xlab = "",
     cex.lab = 1.5,
     cex.axis = 1.5,
     cex.main = 2)
```



```
cutree(sampleTree4, k = 2)
```

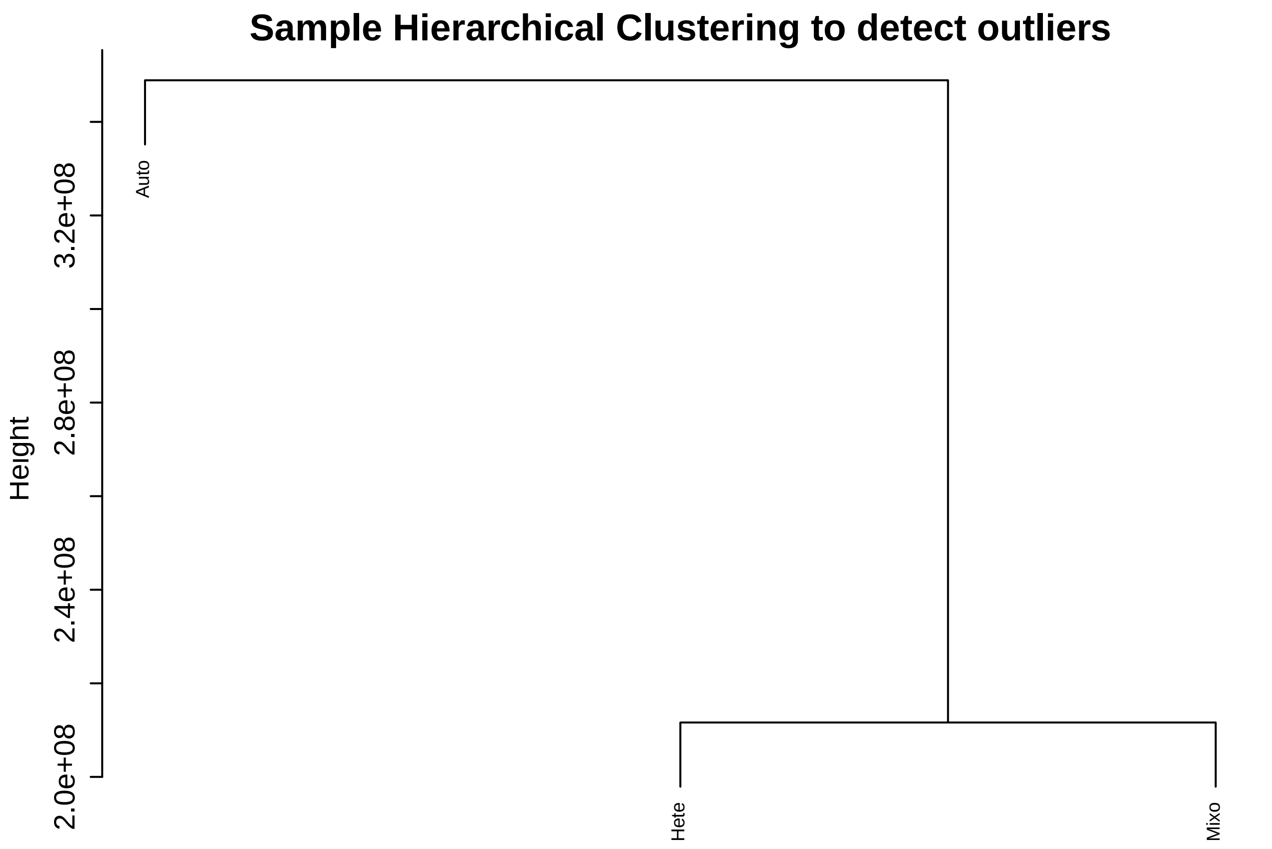
```
## LFQ.intensity.HeteTP10R2 LFQ.intensity.HeteTP10R3  LFQ.intensity.MixoTP6R1
##                        1                        1                        2
##  LFQ.intensity.MixoTP6R2
##                        2
```

The last result seemed somewhat better than the previous ones, since we could in fact properly cluster the samples.

We will try to improve these results by taking the median value among the triplicates as the best estimative of the true LFQIntensity value.

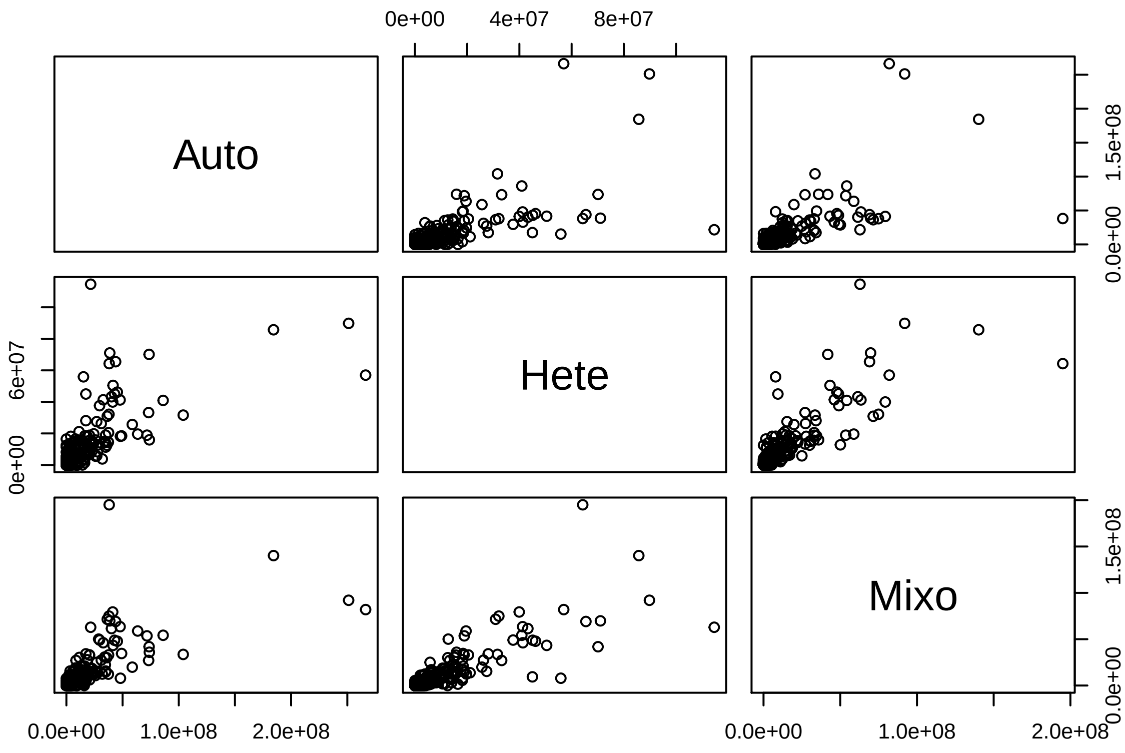
```
# we already took the median value in the previous report,
# so now we only load the data and redo the clustering

valid_proteins_long <- read.csv("proteins.csv", sep = ",", header = TRUE)
valid_proteins_wide <- valid_proteins_long %>% select(-Time) %>%
  spread(Growth, MedLFQ)

# removes the ID column, transposes the valid_proteins_wide dataframe and
# then it turns the matrix back to a data.frame class object
datExprA2 = as.data.frame(t(valid_proteins_wide[, -c(1:1)]))
```

```r
# picks the protein ids from valid_proteins_wide ID column and
# stores it as column names in the datExprA2 data.frame
names(datExprA2) = valid_proteins_wide$ID

# picks the triplicate names stored in the valid_proteins_wide columns and
# saves them as row names from datExprA2
rownames(datExprA2) = names(valid_proteins_wide)[-c(1:1)]

# Clustering dendrogram of samples based on their Euclidean distance
sampleTree5 = hclust(dist(datExprA2), method = "average");

# pdf(file = "sampleClustering.pdf", width = 12, height = 9);
par(cex = 0.6);
par(mar = c(0,4,2,0))
plot(sampleTree5,
     main = "Sample Hierarchical Clustering to detect outliers",
     sub = "",
     xlab = "",
     cex.lab = 1.5,
     cex.axis = 1.5,
     cex.main = 2)
```

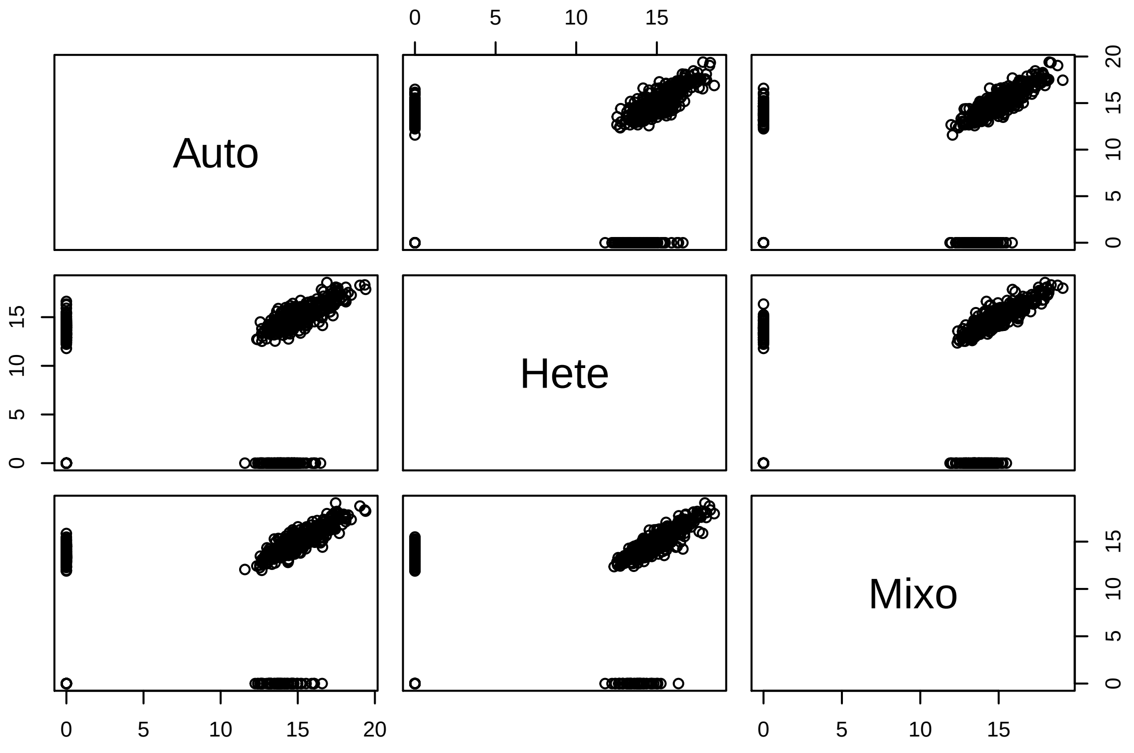


The plot raises the question: Are the Mixotrophic and Heterotrophic growth conditions more similar to each other than to the Autotrophic one?

```r
pairs(valid_proteins_wide[, -1])
```



```r
pairs(log(valid_proteins_wide[, -1] + 1))
```



Scale Free Topology Model Fit

The `WGCNA package` requires us to pick a soft threshold power to estimate both *model fit* and *mean connectivity*, so we can continue with the network construction.

First, we will work with the `datExprA` data.frame devired from the `clean_proteins` data.frame, which is a cleaned version from the original `Proteins` dataset.

```
# 2.a Automatic network construction and module detection

# Choose a set of soft-thresholding powers
powers = c(c(1:10), seq(from = 12, to = 40, by = 2))

# Call the network topology analysis function
sft = pickSoftThreshold(datExprA, powerVector = powers, verbose = 5)
```

```
## pickSoftThreshold: will use block size 757.
##  pickSoftThreshold: calculating connectivity for given powers...
##    ..working on genes 1 through 757 of 757
```

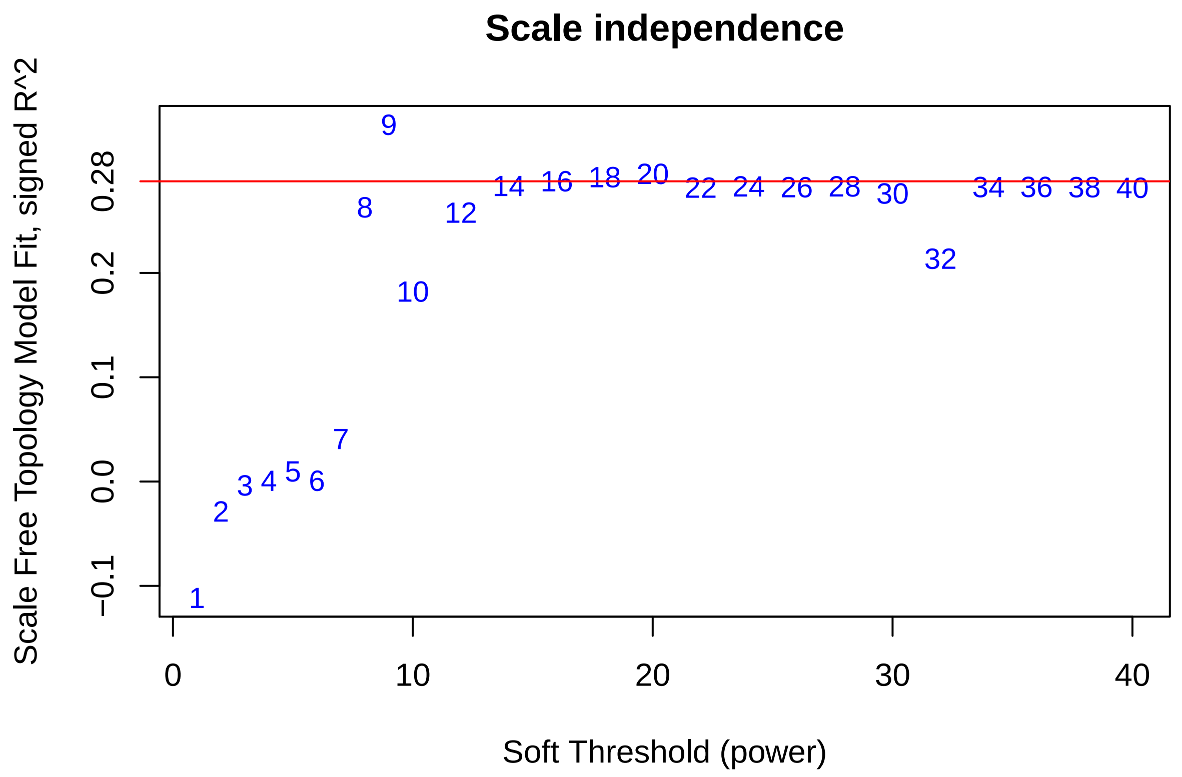
```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
##    Power SFT.R.sq   slope truncated.R.sq mean.k. median.k. max.k.
## 1      1 0.111000  1.8700         0.7800  251.00   255.000  334.0
## 2      2 0.028500  0.4610         0.8300  122.00   122.000  191.0
## 3      3 0.003800  0.1110         0.9180   71.60    69.400  124.0
## 4      4 0.000919 -0.0454         0.8880   47.60    47.900   88.4
## 5      5 0.009460 -0.1200         0.8220   34.40    34.600   66.6
## 6      6 0.000677 -0.0227         0.7630   26.30    25.600   52.1
## 7      7 0.040800 -0.1530         0.4600   21.00    19.900   44.0
## 8      8 0.263000 -0.4830         0.4940   17.40    15.700   41.0
## 9      9 0.342000 -0.9660         0.3260   14.70    12.800   40.5
## 10    10 0.182000 -2.9500         0.0688   12.80    10.400   40.1
## 11    12 0.258000 -4.4300         0.1390   10.10     7.470   39.7
## 12    14 0.284000 -5.1500         0.0931    8.49     5.380   39.5
## 13    16 0.288000 -5.0500         0.0988    7.37     4.040   39.3
## 14    18 0.292000 -4.9400         0.1040    6.59     3.130   39.2
## 15    20 0.295000 -4.8000         0.1090    6.02     2.430   39.2
## 16    22 0.282000 -4.9000         0.0789    5.59     1.910   39.1
## 17    24 0.283000 -4.7800         0.0798    5.26     1.520   39.1
## 18    26 0.282000 -4.6600         0.0782    5.00     1.250   39.1
## 19    28 0.283000 -4.5100         0.0789    4.79     1.020   39.0
## 20    30 0.276000 -4.2900         0.0718    4.62     0.859   39.0
## 21    32 0.214000 -3.8700         0.0247    4.48     0.729   39.0
## 22    34 0.282000 -4.1600         0.0784    4.36     0.633   39.0
## 23    36 0.282000 -4.0700         0.0790    4.27     0.571   39.0
## 24    38 0.282000 -3.9900         0.0792    4.18     0.494   39.0
## 25    40 0.282000 -3.9300         0.0788    4.11     0.423   39.0
```

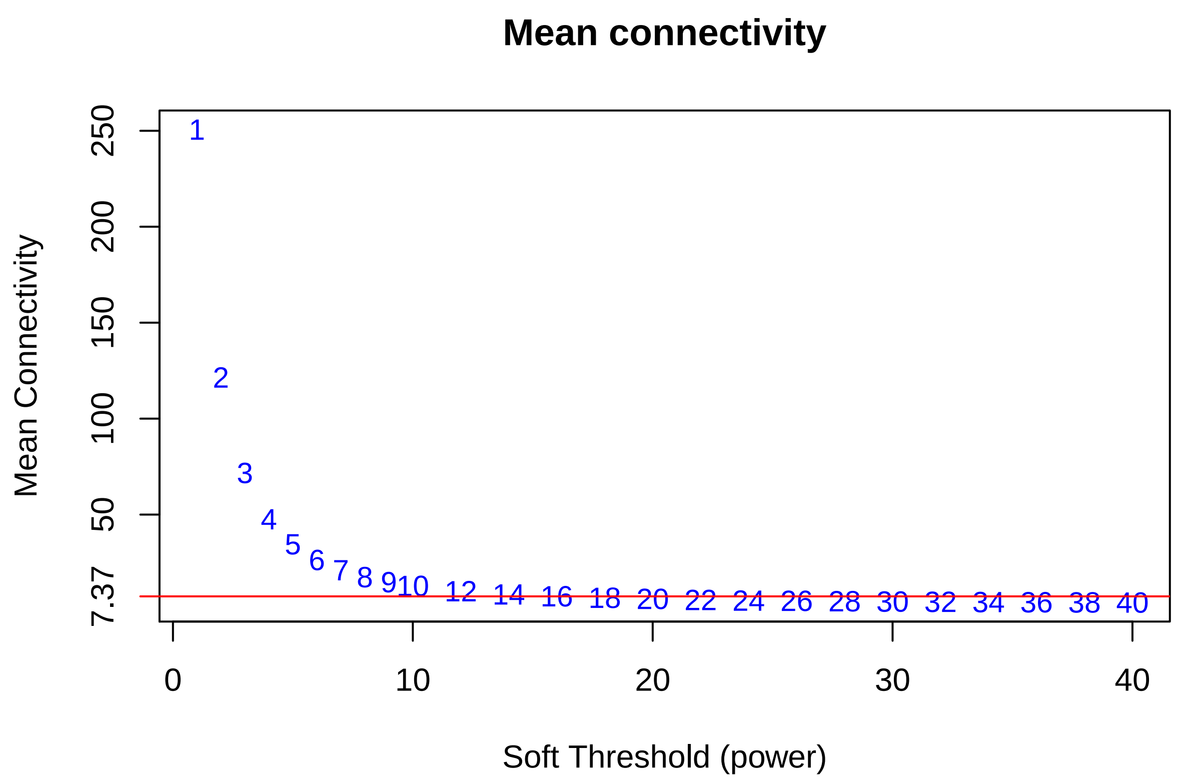
```
# Plot the results:
cex1 = 0.9;

# We try to grab an automated power estimate from the sft list
sft$powerEstimate
```

```
## [1] NA
```

```r
# sft$powerEstimate is the lowest power for which the scale free topology fit
# R^2 exceeds RsquaredCut. If R^2 is below RsquaredCut for all powers, NA is returned.

# Since the function pickSoftThreshold didn't give us a value in
# the sft$powerEstimate, we have to find by ourselves a value
# that is the lowest power for which the scale-free topology fit index curve
# flattens out upon reaching a high value

model_fit <- -sign(sft$fitIndices[ , "slope"]) * sft$fitIndices[ , "SFT.R.sq"]
mean_con <- sft$fitIndices[ , "mean.k."]

# Scale-free topology fit index as a function of the soft-thresholding power
plot(powers,
     model_fit,
     xlab = "Soft Threshold (power)",
     ylab = "Scale Free Topology Model Fit, signed R^2",
     type = "n",
     main = paste("Scale independence"),
     yaxt = "n")

text(powers,
     model_fit,
     labels = powers,
     cex = cex1,
     col = "blue")

axis(2, at = seq(-0.1, 0.2, 0.1))

# Model Fit value corresponding to power 16
abline(h = model_fit[13], col = "red")
axis(2, at = 0.2878373, labels = c(0.28), col = "red")
```



```
# Mean connectivity as a function of the soft-thresholding power
plot(powers,
     mean_con,
     xlab = "Soft Threshold (power)",
     ylab = "Mean Connectivity",
     type = "n",
     main = paste("Mean connectivity"),
     yaxt = "n")

text(powers,
     mean_con,
     labels = powers,
     cex = cex1,
     col = "blue")

axis(2, at = seq(50, 250, 50))

# Mean Connectivity value corresponding to power 16
abline(h = mean_con[13], col = "red")
axis(2, at = 7.370472, labels = c(7.37), col = "red")
```



As can be seen in the two previous plots we have chosen by visual analysis the Soft Treshold Power 16. The Power 16 gives us a 0.2878373 value for the Model Fit Index.

Now, we will do the same for the `datExprA2` data.frame which is based on the median value among triplicates. Let's take a look:

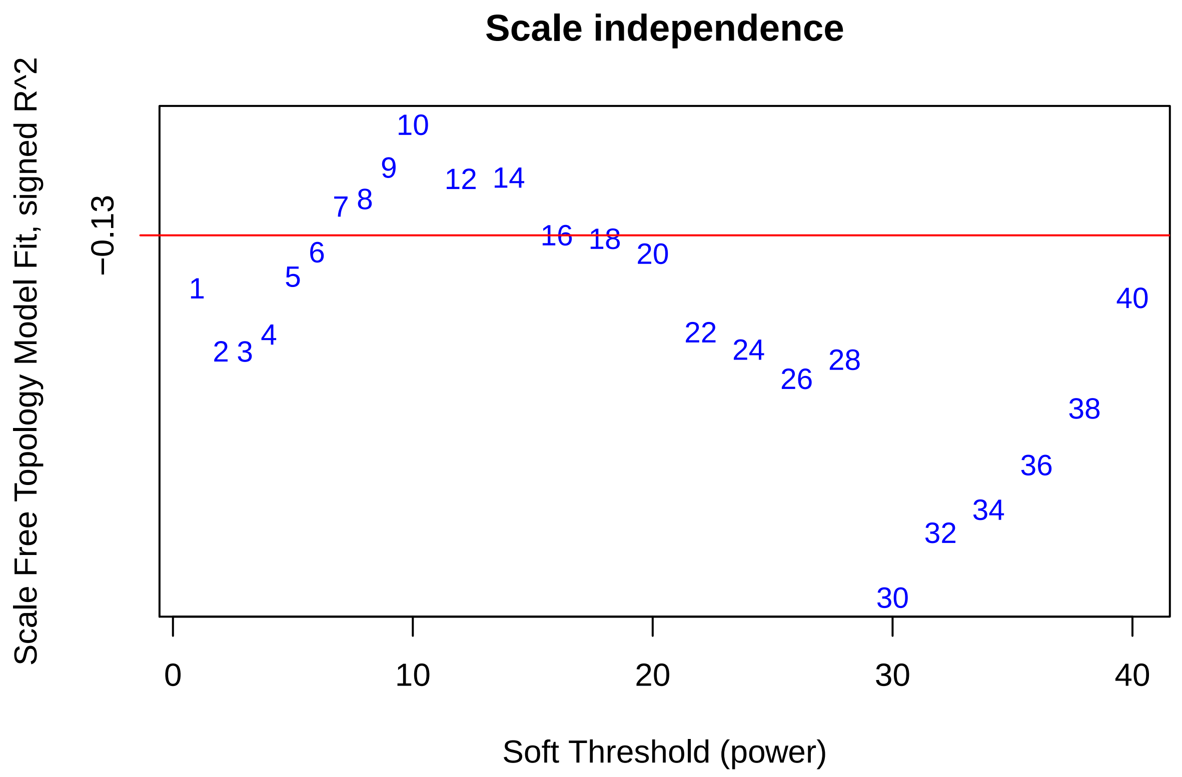
```
# 2.a Automatic network construction and module detection

# Choose a set of soft-thresholding powers
powers = c(c(1:10), seq(from = 12, to = 40, by = 2))

# Call the network topology analysis function
sft2 = pickSoftThreshold(datExprA2, powerVector = powers, verbose = 5)
```

```
## pickSoftThreshold: will use block size 600.
##  pickSoftThreshold: calculating connectivity for given powers...
##    ..working on genes 1 through 600 of 600
##    Power SFT.R.sq slope truncated.R.sq mean.k. median.k. max.k.
## 1      1   0.1800 4.780          0.816   395.0     410.0    433
## 2      2   0.2390 2.640          0.753   313.0     328.0    364
## 3      3   0.2390 2.050          0.625   268.0     282.0    321
## 4      4   0.2230 1.860          0.776   238.0     253.0    289
## 5      5   0.1700 1.770          0.818   217.0     233.0    264
## 6      6   0.1470 1.750          0.879   201.0     219.0    244
## 7      7   0.1050 1.570          0.894   189.0     207.0    229
## 8      8   0.0976 1.550          0.849   179.0     198.0    217
## 9      9   0.0684 1.280          0.785   170.0     191.0    208
## 10    10   0.0287 0.769          0.687   163.0     184.0    201
## 11    12   0.0787 1.160          0.590   151.0     170.0    189
## 12    14   0.0775 1.020          0.497   141.0     158.0    180
## 13    16   0.1310 1.190          0.459   133.0     148.0    172
## 14    18   0.1350 1.190          0.386   127.0      139.0     166
## 15    20   0.1480 1.020          0.286   121.0      131.0     160
## 16    22   0.2210 1.000          0.357   117.0      125.0     155
## 17    24   0.2370 0.954          0.360   112.0      119.0     151
## 18    26   0.2650 0.907          0.378   109.0      114.0     148
## 19    28   0.2470 0.864          0.332   105.0      109.0     144
## 20    30   0.4680 0.898          0.600   102.0      105.0     141
## 21    32   0.4070 0.872          0.492    99.6      101.0     138
## 22    34   0.3860 0.845          0.460    97.1       97.8     136
## 23    36   0.3450 0.790          0.394    94.8       94.9     134
## 24    38   0.2920 0.744          0.313    92.6       92.3     132
## 25    40   0.1890 0.612          0.125    90.7       89.7     130
```

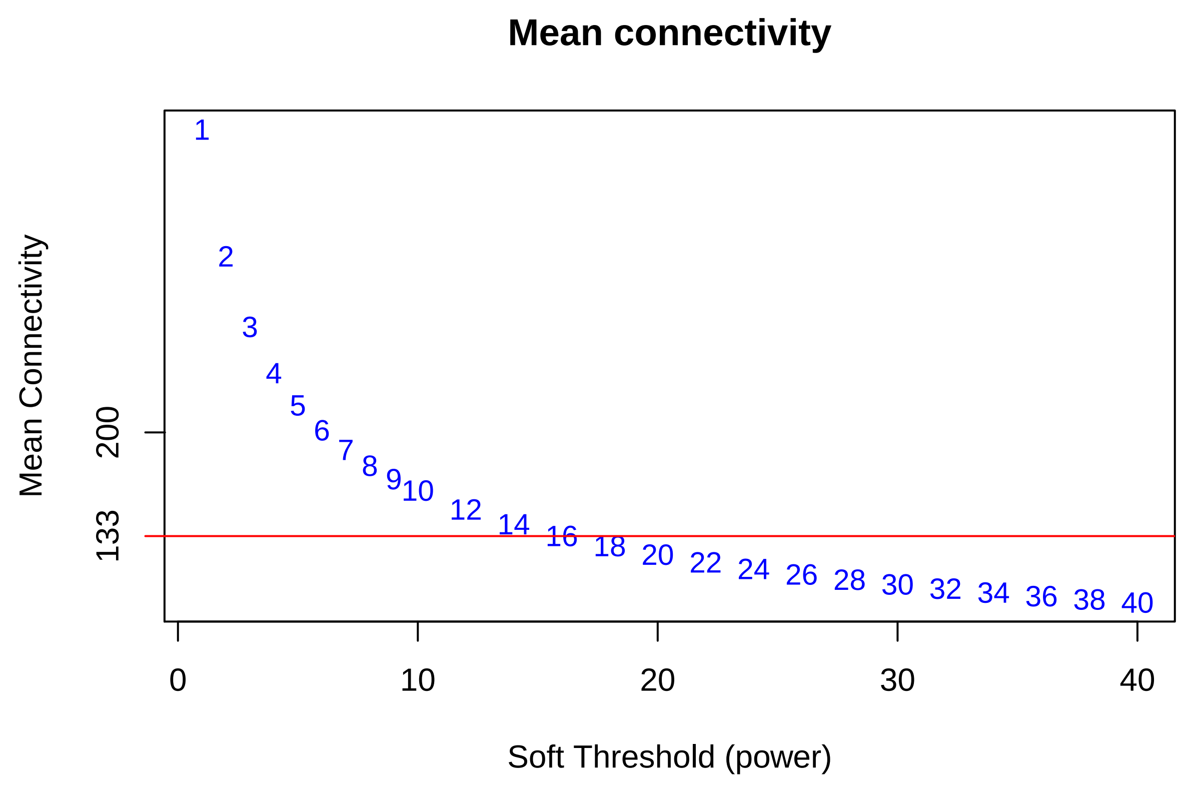
```
# Plot the results:
cex1 = 0.9;

# We try to grab an automated power estimate from the sft2 list
sft2$powerEstimate
```

```
## [1] NA
```

```
# sft2$powerEstimate is the lowest power for which the scale free topology fit
# R^2 exceeds RsquaredCut. If R^2 is below RsquaredCut for all powers, NA is returned.

# Since the function pickSoftThreshold didn't give us a value in
# the sft2$powerEstimate, we have to find by ourselves a value
# that is the lowest power for which the scale-free topology fit index curve
# flattens out upon reaching a high value

model_fit2 <- -sign(sft2$fitIndices[ , "slope"]) * sft2$fitIndices[ , "SFT.R.sq"]
mean_con2 <- sft2$fitIndices[, "mean.k."]

# Scale-free topology fit index as a function of the soft-thresholding power
plot(powers,
     model_fit2,
     xlab = "Soft Threshold (power)",
     ylab = "Scale Free Topology Model Fit, signed R^2",
     type = "n",
     main = paste("Scale independence"),
     yaxt = "n")

text(powers,
     model_fit2,
     labels = powers,
     cex = cex1,
     col = "blue")

#axis(2, at = seq(-0.1, 0.2, 0.1))

# Model Fit value corresponding to power 16
abline(h = model_fit2[13], col = "red")
axis(2, at = -0.1312341, labels = c(-0.13), col = "red")
```



```r
# Mean connectivity as a function of the soft-thresholding power
plot(powers,
     mean_con2,
     xlab = "Soft Threshold (power)",
     ylab = "Mean Connectivity",
     type = "n",
     main = paste("Mean connectivity"),
     yaxt = "n")

text(powers,
     mean_con2,
     labels = powers,
     cex = cex1,
     col = "blue")

axis(2, at = c(200))

# Mean Connectivity value corresponding to power 16
abline(h = mean_con2[13], col = "red")
axis(2, at = 133.4083, labels = c(133), col = "red")
```



Since we couldn't find a good soft-thresholding power for the `datExprA2` data.frame we decided to use the power 16 found for the `datExprA` data.frame. Unfortunately, it gave us a worse fit both graphically and numerically, the Model Fit Index this time was -0.1312341. The results on both data frames were very much influenced by the minimum requirements of `WGCNA` package, according to the WGCNA FAQ one needs at least 15 samples to find results that are not too noisy, it also recommends at least 20 samples for better results.

Saving our Expression Data

Finally we save our data frames `datExprA` and `datExprA2` for further analysis.

```
write.csv(datExprA, file = "datExprA.csv", row.names = TRUE)
datExprA <- read.csv("datExprA.csv", sep = ",", header = TRUE)
rownames(datExprA) = datExprA$X
datExprA <- datExprA[ , -c(1)]

write.csv(datExprA2, file = "datExprA2.csv", row.names = TRUE)
datExprA2 <- read.csv("datExprA2.csv", sep = ",", header = TRUE)
rownames(datExprA2) = datExprA2$X
datExprA2 <- datExprA2[ , -c(1)]
```