Adjacency, TOM Matrices & Rand Index

Filipe Russo

June 02, 2019

Adjacency Matrix

The `adjacency` function from the `R` package `WGCNA` returns a square *adjacency matrix* for the network based on correlation or distance, it expects the mandatory arguments `datExpr`, `type` and `power`, but also has several optional ones.

The `datExpr` argument expects a data frame containing expression data, where columns correspond to genes (proteins in our case) and rows to samples. The `type` argument is the type of the network and expects one of the following strings: unsigned, signed, signed hybrid, distance. Finally, the `power` argument is the soft thresholding power explored in the previous report.

In the equations below we will show the mathematical formulation of each option for the `type` argument, in them the `cor` function from base R is used to calculate Pearson Correlation.

For `type = "unsigned"` we have the following equation:

$$adjacency = |cor|^{power}$$

For `type = "signed"` we have:

$$adjacency = (0.5 * (1 + cor))^{power}$$

For `type = "signed hybrid"` we have:

if cor > 0:

$$adjacency = cor^{power}$$

and otherwise:

$$adjacency = 0$$

Finally, for `type = "distance"` we have the equation:

$$adjacency = (1 - (dist/max(dist))^2)^{power}$$

We want a network where positively and negatively coexpressed proteins are both well contemplated. Such is not the case for `type = "signed"`, because here nodes with negative correlation are considered unconnected. On the other hand, for `type = "unsigned"` the absolute value of the correlation is taken into account, so negatively coexpressed proteins are treated exactly as positively coexpressed ones and their particularities not fully contemplated.

Topological Overlap Matrix

The `TOMSimilarity` function from the `R` package `WGCNA` calculates the *topological overlap matrix* from a given *adjacency matrix*, it expects the mandatory arguments `adjMat` and `TOMType`, but also has several optional ones.

The `adjMat` argument expects a square symmetric matrix with entries between 0 and 1 (negative values are allowed if `TOMType = "signed"`).

The `TOMType` argument expects one of the following strings: unsigned, signed or signed Nowick. The `TOMType = "signed"` takes into account possible anti-reinforcing connection strengths.

One can easily recover the sign of the correlation back to the result of the `adjacency` function as follows:

$$\tilde{a}_{ij} = a_{ij} \times sign(cor(x_i, x_j))$$

The signed TOM is then defined as:

$$TOM_{ij}^{signed} = \frac{\left| a_{ij} + \sum_{u \neq i,j} \tilde{a}_{iu}\,\tilde{a}_{uj} \right|}{min(k_i, k_j) + 1 - |a_{ij}|}$$

Where k_i and k_j denote the connectivity of nodes i and j:

$$k_i = \sum_{u \neq i} |\tilde{a}_{ui}|$$

In contrast, unsigned TOM is defined as follows (note the difference in the placement of absolute values in the numerator):

$$TOM_{ij} = \frac{|a_{ij}| + \sum_{u \neq i,j} |\tilde{a}_{iu}\,\tilde{a}_{uj}|}{min(k_i, k_j) + 1 - |a_{ij}|}$$

At this time we will be passing to the `TOMsimilarity` function simply the result from the `adjacency` function times the sign of the correlations.

WGCNA

We pass our protein expression data to the `adjancency()` function with `power = 16` and `type = "unsigned"`, then we update the returned `adjancency` matrix with the sign of the original Pearson Correlation so to have the necessary signed `adjMat` matrix. This one is then passed to the `TOMSimilarity()` function, which returns the `TOM` matrix.

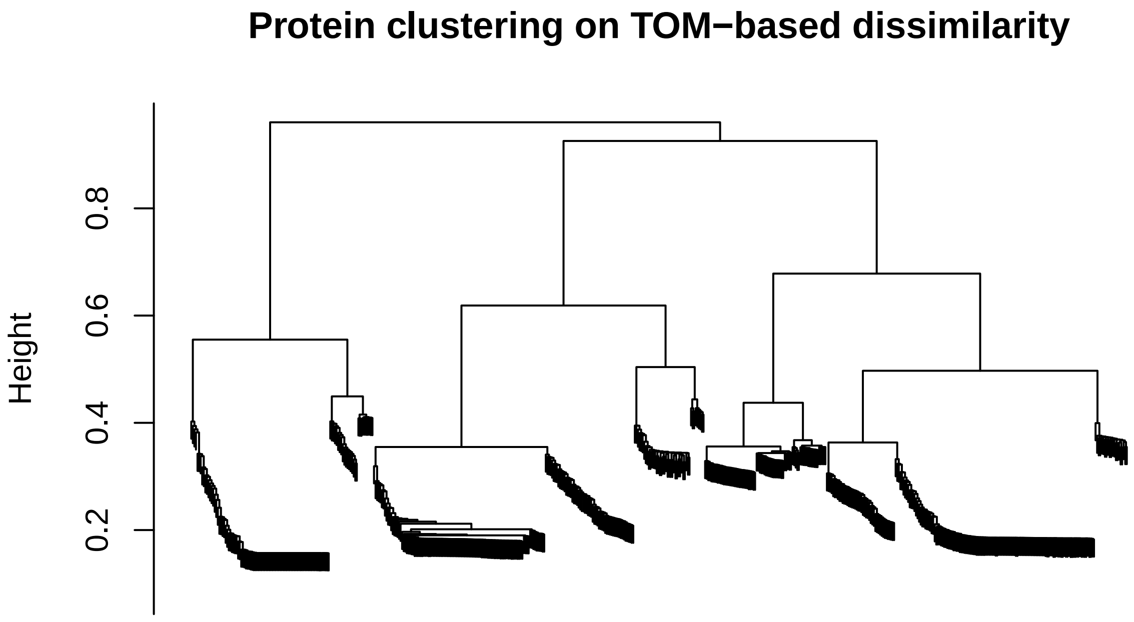
```r
library(WGCNA)
```

```r
datExprA <- read.csv("datExprA2.csv", sep = ",", header = TRUE)
rownames(datExprA) = datExprA$X
datExprA <- datExprA[ , -c(1)]

adjacency = adjacency(datExpr = datExprA, type = "unsigned", power = 16)
adjMat = adjacency * sign(cor(datExprA))
```

```r
# Turns adjMat matrix into TOM matrix
TOM = TOMsimilarity(adjMat = adjMat, TOMType = "signed")
```

```
## ..connectivity..
## ..matrix multiplication (system BLAS)..
## ..normalization..
## ..done.
```

```r
dissTOM = 1 - TOM

# Call the hierarchical clustering function
geneTree = hclust(as.dist(dissTOM), method = "average")
geneTree$labels = names(datExprA)

# Plot the resulting clustering tree (dendrogram)
plot(geneTree, xlab = "", sub = "",
     main = "Protein clustering on TOM-based dissimilarity",
     labels = FALSE, hang = 0.04)
```



By now we already have a clustering based on TOM dissimilarity: (1 - TOM). We decide our modules must have a minimum cluster size of 10 Proteins and go on with the analysis.

```r
# We like large modules, so we set the minimum module size relatively high:
minModuleSize = 10

# Module identification using dynamic tree cut:
dynamicMods = cutreeDynamic(dendro = geneTree,
                            distM = dissTOM,
                            deepSplit = 2,
                            pamRespectsDendro = FALSE,
                            minClusterSize = minModuleSize)
```

```
##  ..cutHeight not given, setting it to 0.952  ===>  99% of the (truncated) height range in dendro.
##  ..done.
```

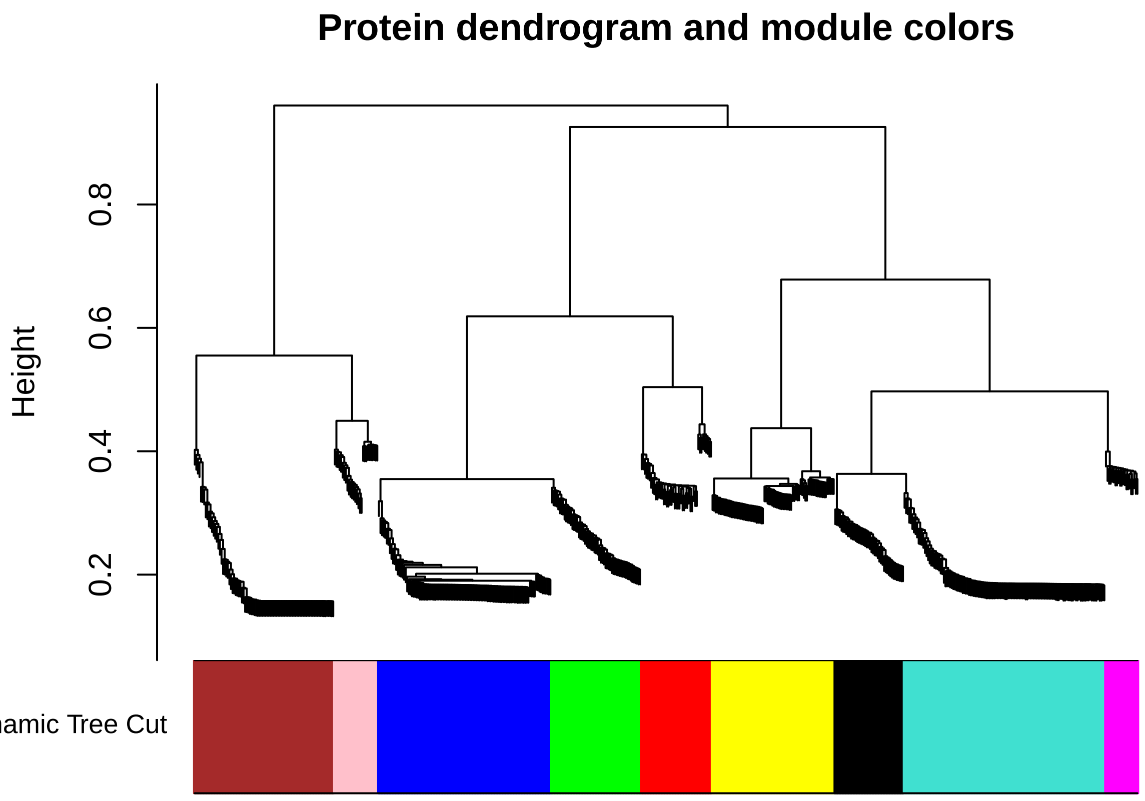
```
table(dynamicMods)
```

```
## dynamicMods
##   1   2   3   4   5   6   7   8   9
## 128 110  89  78  57  45  44  28  21
```

```
# Convert numeric lables into colors
dynamicColors = labels2colors(dynamicMods)
table(dynamicColors)
```

```
## dynamicColors
##     black      blue     brown     green   magenta      pink       red
##        44       110        89        57        21        28        45
## turquoise    yellow
##       128        78
```

```
# Plot the dendrogram and colors underneath
plotDendroAndColors(geneTree,
                    dynamicColors,
                    "Dynamic Tree Cut",
                    dendroLabels = FALSE,
                    hang = 0.03,
                    addguide = TRUE,
                    guideHang = 0.05,
                    main = "Protein dendrogram and module colors")
```

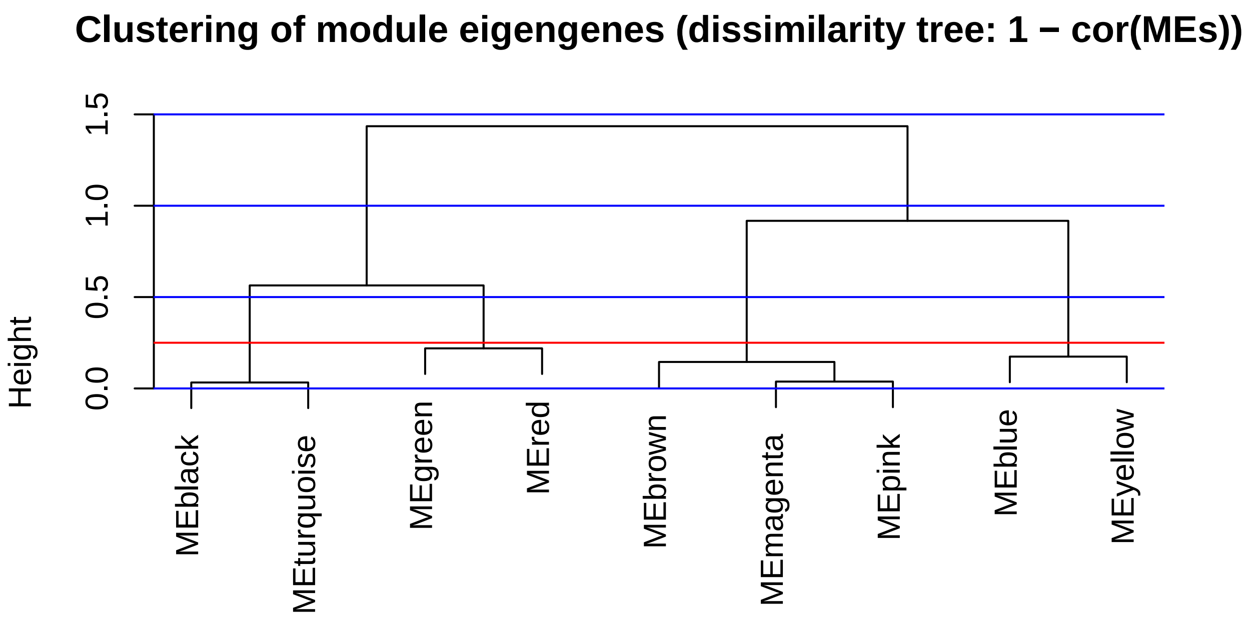


```
Partition_A <- dynamicColors
```

The `cutreeDynamic` function returns our 600 proteins in a 9 modules partition, which we stored in the `Partition_A` variable. At the end of the report this partition will be stored in a new column of our proteins dataframe, which we will save in an external csv file for future analysis.

Now we will merge modules with an intermodule correlation of at least 0.75.

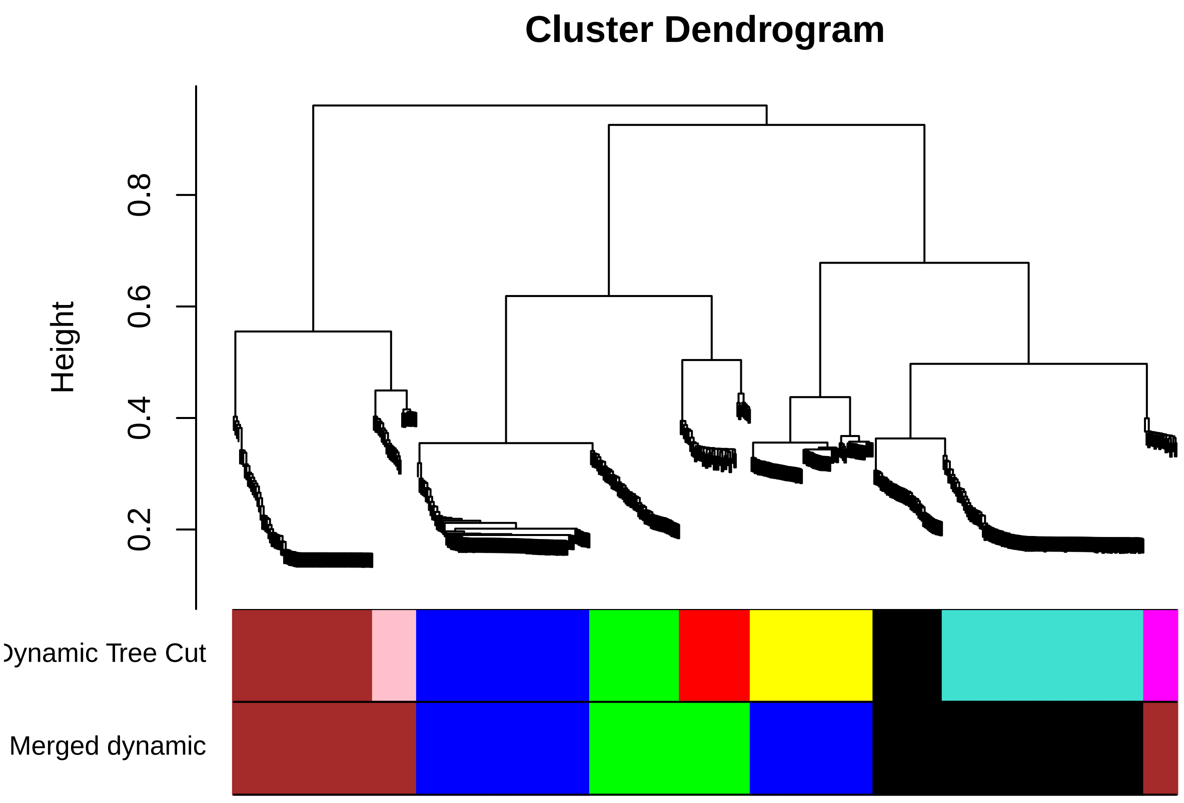
```r
# Calculate eigengenes
MEList = moduleEigengenes(datExprA, colors = dynamicColors)
MEs = MEList$eigengenes

# Calculate dissimilarity of module eigengenes
MEDiss = 1 - cor(MEs);

# Cluster module eigengenes
METree = hclust(as.dist(MEDiss), method = "average")

# Plot the result
plot(METree,
     main =
     "Clustering of module eigengenes (dissimilarity tree: 1 - cor(MEs))",
     xlab = "",
     sub = "")

# Correlation of at least 0.75 necessary to merge modules
MEDissThres = 0.25

# Plot the cut line into the dendrogram
abline(h = MEDissThres, col = "red")
abline(h = 2, col = "blue")
abline(h = 1.5, col = "blue")
abline(h = 1, col = "blue")
abline(h = 0.5, col = "blue")
abline(h = 0, col = "blue")
```



```r
# Call an automatic merging function
merge = mergeCloseModules(datExprA,
                          dynamicColors,
                          cutHeight = MEDissThres,
                              verbose = 3)
```

```
##  mergeCloseModules: Merging modules whose distance is less than 0.25
##    multiSetMEs: Calculating module MEs.
##      Working on set 1 ...
##      moduleEigengenes: Calculating 9 module eigengenes in given set.
##    multiSetMEs: Calculating module MEs.
##      Working on set 1 ...
##      moduleEigengenes: Calculating 4 module eigengenes in given set.
##    Calculating new MEs...
##    multiSetMEs: Calculating module MEs.
##      Working on set 1 ...
##      moduleEigengenes: Calculating 4 module eigengenes in given set.
```

```
# The merged module colors
mergedColors = merge$colors

# Eigengenes of the new merged modules:
mergedMEs = merge$newMEs

# Plot the comparision between Dynamic Tree Cut and Merged Dynamic
plotDendroAndColors(geneTree,
                    cbind(dynamicColors, mergedColors),
                    c("Dynamic Tree Cut", "Merged dynamic"),
                    dendroLabels = FALSE,
                    hang = 0.03,
                    addguide = TRUE,
                    guideHang = 0.05)
```



```r
# Rename to moduleColors
moduleColors = mergedColors
table(mergedColors)
```

```
## mergedColors
## black  blue brown green
##   172   188   138   102
```

```r
# Construct numerical labels corresponding to the colors
colorOrder = c("grey", standardColors(50))
moduleLabelsDynamic = match(dynamicColors, colorOrder) - 1
MEs = mergedMEs

moduleLabelsMerged = match(mergedColors, colorOrder) - 1

Partition_B <- mergedColors
```

The `mergeCloseModules` function with `cutHeight = MEDissThres`, where MEDissThres is set to 0.25, turns our 9 modules partition into a 4 modules partition, where all modules have an intermodule correlation less than 0.75. We stored this second partition in the `Partition_B` variable.

Rand Index

The Rand Index in statistics, and in particular in data clustering, is a measure of the similarity between two data clusterings.

Let $\mathcal{P}$ and $\mathcal{L}$ be two partitions of a given set $\mathcal{S}$.

Consider all possible pairs of elements of the partitions, so to define a, b, c and d as follows:

a : number of pairs in the same group in both partitions.

b : number of pairs in the same group in partition $\mathcal{P}$ but not in partition $\mathcal{L}$.

c : number of pairs in the different groups in partition $\mathcal{P}$ but not in partition $\mathcal{L}$.

d : number of pairs in different groups in both partitions.

Then, the Rand Index is defined by:

$$RAND = \frac{a + d}{a + b + c + d}$$

where N : is the number of elements in the set $\mathcal{S}$.

According to the paper "Properties of the Hubert–Arabie Adjusted Rand Index" by Douglas Steinley (2004), among the indices $RAND$, $Jaccard$, $Fowlkes\,and\,Mallows$, ARI_{MA} and ARI_{HA}, the ARI_{HA} (Hubert–Arabie Adjusted Rand Index) is the most robust.

$$ARI_{HA} = \frac{\binom{N}{2}(a+d) - [(a+b)(a+c)+(c+d)(b+d)]}{\binom{N}{2}^2 - [(a+b)(a+c)+(c+d)(b+d)]}$$

Below we apply the `adjustedRand()` function from the `R` package `clues` to check the similarity between our `Partition_A` and `Partition_B`, we are actually passing their numerically labeled counterparts, the variables `moduleLabelsDynamic` and `moduleLabelsMerged` respectively, instead of the color labeled ones because the `adjustedRand()` function expects the input this way.

```r
library(clues)
adjustedRand(moduleLabelsDynamic, moduleLabelsMerged)
```

```
##      Rand        HA        MA        FM   Jaccard
## 0.8790985 0.6312582 0.6336256 0.7325575 0.5366405
```

The researcher Douglas Steinly offers the following heuristics to check the quality of the cluster recovery:

- values greater than 0.90 can be viewed as excellent recovery
- values greater than 0.80 can be considered good recovery
- values greater than 0.65 can be considered moderate recovery
- values less tgan 0.65 reflect poor recovery

Our Hubert–Arabie Adjusted Rand Index was 0.6312582, which according to our reference paper means a poor recovery. So the `Partition_A` comprised of the 600 proteins grouped in 9 modules can be consired very different from the `Partition_B` comprised of the 600 proteins grouped in 4 modules.

Saving the Data

Finally, we store our `Partition_A` and `Partiton_B` variable in a new **`proteins`** dataframe, which we save for further analysis.

```r
library(dplyr)
proteins <- as.data.frame(t(datExprA))
proteins <- proteins %>% mutate(ID = rownames(proteins),
                                Partition_A = Partition_A,
                                Partition_B = Partition_B)
write.csv(proteins, file = "proteins.csv", row.names = FALSE)
proteins <- read.csv("proteins.csv", sep = ",", header = TRUE)
```