WGCNA_TP

November 14, 2018

1 TP WGCNA

1.1 M2BI Biologie de Systemes

1.1.1 Costas Bouyioukos 2018

1.2 Part 0

Settingup the environment

```
In [2]: # Load WGCNA package
        library(WGCNA)
        # The following setting is important to import the data properly.
        options(stringsAsFactors = FALSE)
```

```
Loading required package: dynamicTreeCut
Loading required package: fastcluster

Attaching package: fastcluster

The following object is masked from package:stats:

    hclust
```

```
==========================================================================
*
*  Package WGCNA 1.66 loaded.
*
*    Important note: It appears that your system supports multi-threading,
*    but it is not enabled within WGCNA in R.
*    To allow multi-threading within WGCNA with all available cores, use
*
*          allowWGCNAThreads()
*
*    within R. Use disableWGCNAThreads() to disable threading if necessary.
*    Alternatively, set the following environment variable on your system:
```

```
*
*          ALLOW_WGCNA_THREADS=<number_of_processors>
*
*    for example
*
*          ALLOW_WGCNA_THREADS=8
*
*    To set the environment variable in linux bash shell, type
*
*           export ALLOW_WGCNA_THREADS=8
*
*     before running R. Other operating systems or shells will
*     have a similar command to achieve the same aim.
*
========================================================================
```

```
Attaching package: WGCNA

The following object is masked from package:stats:

    cor
```

1.3 Part 1

Loading of Gene Expression data.

The files contain simulated gene expression data, are provided in the zip file and have the proper filenames.

```
In [4]: datGeneSummary=read.csv("GeneSummaryTutorial.csv")
        datTraits=read.csv("TraitsTutorial.csv")
        datGeneExpr=read.csv("MicroarrayDataTutorial.csv")
```

We can take a quick look, they should look like the foloowing.

```
In [5]: head(datGeneExpr)
```

GeneName	Sample1	Sample2	Sample3	Sample4	Sample5	Sample6	Sample7	Sa
Gene1	-0.76903180	0.2010896	-0.8929247	0.28616899	-0.6224324	1.569568	1.4352953	1.1
Gene2	-0.25281902	0.4293141	-1.1007358	0.05983439	-0.6152275	1.669728	1.2668673	0.6
Gene3	0.33073345	-0.2741976	0.7029428	0.02233195	0.8093945	-1.673001	-0.6530627	-1.
Gene4	0.01335525	0.1155758	-0.4554590	0.04184443	-0.4602345	2.132252	1.1245297	1.1
Gene5	0.30580959	0.1004434	-0.6086998	-0.32522219	-1.0474164	2.914156	0.5821223	0.9
Gene6	0.84871917	-0.8866578	0.4384386	-0.02703332	0.4270420	-2.057772	-0.4740660	-1.

Then we reformat the data and set some appropriate gene names etc.

```
In [11]: # This vector contains the microarray sample names
         SampleName=colnames(datGeneExpr)[2:length(colnames(datGeneExpr))]
         # This vector contains the gene names
         GeneName=datGeneExpr$GeneName
         # We transpose the data so that the rows correspond to samples and the columns corres
         # Since the first column contains the gene names, we exclude it.
         datExpr=data.frame(t(datMicroarrays[,-1]))
         names(datExpr)=datGeneExpr[,1]
         dimnames(datExpr)[[1]]=SampleName
         #Also, since we simulated the data, we keep the true module color:
         truemodule=datGeneSummary$truemodule
```

```
In [13]: head(datExpr)
```

	Gene1	Gene2	Gene3	Gene4	Gene5	Gene6	Gene7	G
Sample1	-0.7690318	-0.25281902	0.33073345	0.01335525	0.3058096	0.84871917	-0.05517512	-
Sample2	0.2010896	0.42931409	-0.27419755	0.11557575	0.1004434	-0.88665785	-0.71263979	0
Sample3	-0.8929247	-1.10073581	0.70294278	-0.45545899	-0.6086998	0.43843862	-0.89374789	-
Sample4	0.2861690	0.05983439	0.02233195	0.04184443	-0.3252222	-0.02703332	-0.17070069	0
Sample5	-0.6224324	-0.61522751	0.80939447	-0.46023446	-1.0474164	0.42704199	0.88542282	-
Sample6	1.5695682	1.66972847	-1.67300053	2.13225233	2.9141561	-2.05777234	2.42458573	2

```
In [18]: # First, make sure that the array names in the file datTraits line up with those in t
         table(dimnames(datExpr)[[1]]==datTraits$ArrayName)
```

```
TRUE
  50
```

```
In [28]: # Then, keep the microarray sample trait
         y = datTraits$Trait.y
```

1.4 Part 2

1.4.1 Basic data pre-processing, cleaning etc.

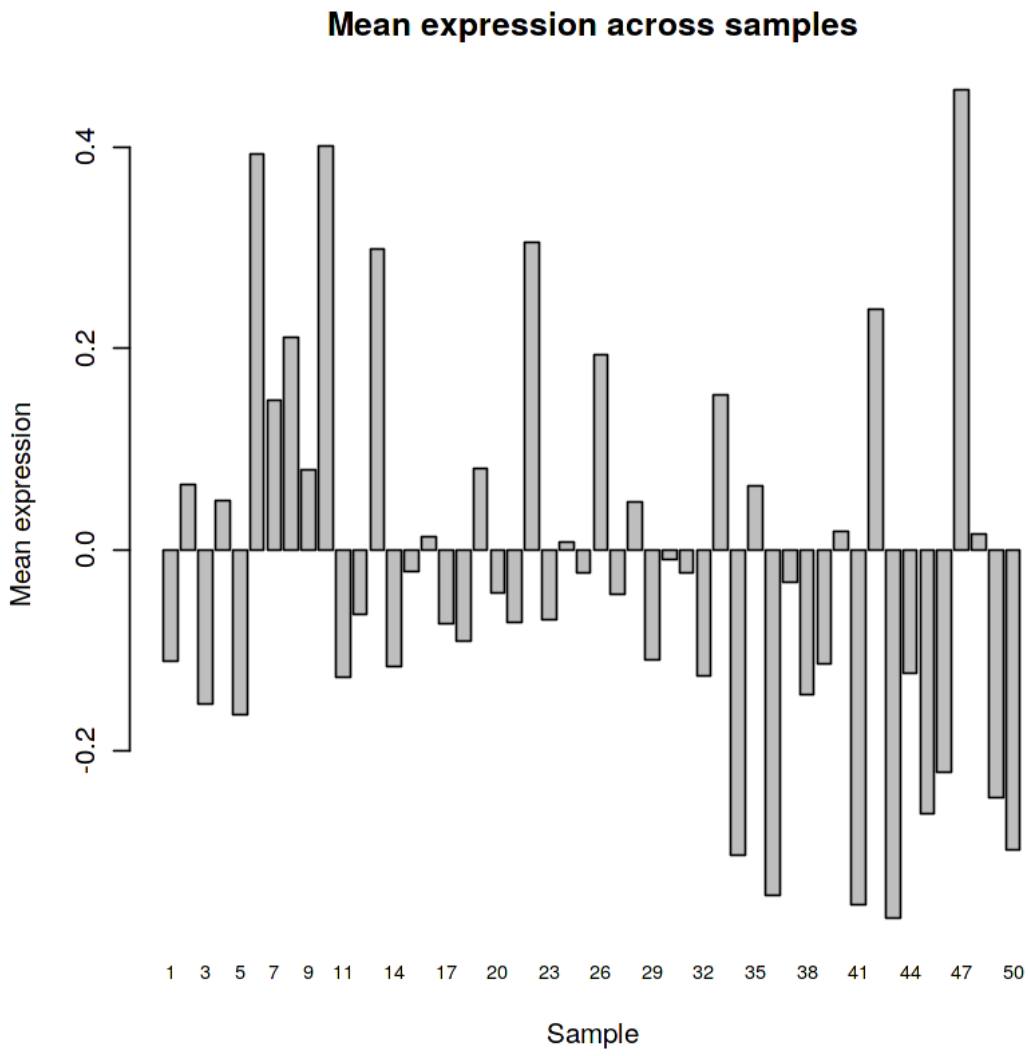
We start by defining the mean expression per sample and the number of missing values.

```
In [20]: meanExpressionBySample=apply(datExpr, 1, mean, na.rm=T)
         NumberMissingBySample=apply(is.na(data.frame(datExpr)), 1, sum)
```

```
In [23]: head(NumberMissingBySample) # NO missing values (this is simulated data).
```

Sample1 0 **Sample2** 0 **Sample3** 0 **Sample4** 0 **Sample5** 0 **Sample6** 0

```
In [25]: # Visualise mean expression per sample
         barplot(meanExpressionBySample,
         xlab = "Sample", ylab = "Mean expression",
         main ="Mean expression across samples",
         names.arg = c(1:50), cex.names = 0.7)
```

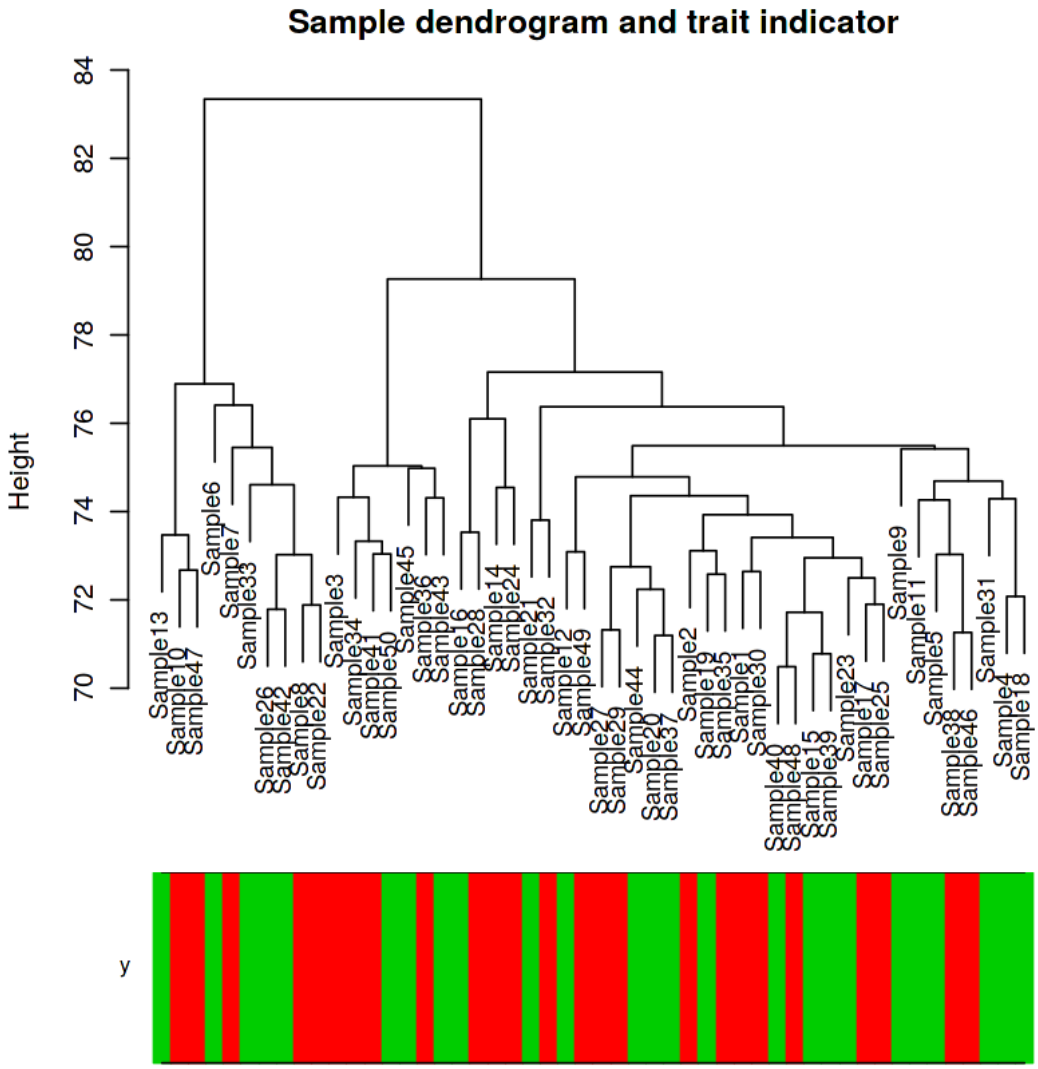


In [26]:
```
# Calculate the variances of the genes and the number of present entries
variancedatExpr=as.vector(apply(as.matrix(datExpr),2,var, na.rm=T))
no.presentdatExpr=as.vector(apply(!is.na(as.matrix(datExpr)),2, sum) )
# Another way of summarizing the number of pressent entries
table(no.presentdatExpr) # simulated data so all genes will be selected.
```

```
no.presentdatExpr
  50
3000
```

1.4.2 Simple way to detect outliers.

In [29]:
```
plotClusterTreeSamples(datExpr=datExpr, y=y)
```



No obvious outlier, but this is an important pre-processing step

(together with PCA) for EVERY Gene Expression analysis that you perform.

1.5 Part 3

Construction of a weighted gene expression network and network modules.

1.5.1 Network modules.

First we construct a co-expression network by soft threshold the correlation matrix with a beta = 6.

```
In [ ]: # here we define the adjacency matrix using soft thresholding with beta=6
        ADJ1=abs(cor(datExpr, use="p"))^6
        # When you have relatively few genes (<5000) use the following code
        k=as.vector(apply(ADJ1, 2, sum, na.rm=T))
        # When you have a lot of genes use the following code
        #k=softConnectivity(datE=datExpr,power=6) # We have 3000.
        # Plot a histogram of k and a scale free topology plot
        hist(k)
        scaleFreePlot(k, main="Check scale free topology")
```

```
In [ ]: # We restrict our analysis to only the top 3600 connected geens.
        datExpr=datExpr[, rank(-k,ties.method="first" )<=3600]
```

1.5.2 Comparing various module detection methods

Clustering dissimilarity from the adjacency matrix directly.

```
In [ ]: # Turn adjacency into a measure of dissimilarity
        dissADJ=1-ADJ1
```

Using topological overlap to define dissimilarity.

```
In [ ]: dissTOM=TOMdist(ADJ1)
```

1.5.3 Average linkage hierachical clustering with adjacency-based dissimilarity

```
In [ ]: hierADJ=hclust(as.dist(dissADJ), method="average" )
        # Plot the resulting clustering tree together with the true color assignment
        plotDendroAndColors(hierADJ, colors = data.frame(truemodule), dendroLabels = FALSE, ha
        main = "Gene hierarchical clustering dendrogram and simulated module colors" )
```

1.5.4 Module definition via fixed height cut-off

```
In [ ]: colorStaticADJ=as.character(cutreeStaticColor(hierADJ, cutHeight=.99, minSize=20))
        # Plot the dendrogram with module colors
        plotDendroAndColors(hierADJ, colors = data.frame(truemodule, colorStaticADJ),
        dendroLabels = FALSE, abHeight = 0.99,
        main = "Gene dendrogram and module colors")
```

1.5.5 Module definition via dynamic branch cutting methods

One method is the "true" method that uses only the dendrogram the second a hybrid method that uses both the dendrogramas well as the dissimilarity matrix.

```
In [ ]: branch.number=cutreeDynamic(hierADJ,method="tree")
        # This function transforms the branch numbers into colors
        colorDynamicADJ=labels2colors(branch.number )
        colorDynamicHybridADJ=labels2colors(cutreeDynamic(hierADJ,distM= dissADJ,
        cutHeight = 0.998, deepSplit=2, pamRespectsDendro = FALSE))
```

```
# Plot results of all module detection methods together:
plotDendroAndColors(dendro = hierADJ,
colors=data.frame(truemodule, colorStaticADJ,
colorDynamicADJ, colorDynamicHybridADJ),
dendroLabels = FALSE, marAll = c(0.2, 8, 2.7, 0.2),
main = "Gene dendrogram and module colours")
```

1.5.6 Module definition using the topological overlap based dissimilarity

We do the same steps as above but by using the dissimilarity from the TOP method.

In [ ]:
```
# Calculate the dendrogram
hierTOM = hclust(as.dist(dissTOM),method="average");
# The reader should vary the height cut-off parameter h1
# (related to the y-axis of dendrogram) in the following
colorStaticTOM = as.character(cutreeStaticColor(hierTOM, cutHeight=.99, minSize=20))
colorDynamicTOM = labels2colors (cutreeDynamic(hierTOM,method="tree"))
colorDynamicHybridTOM = labels2colors(cutreeDynamic(hierTOM, distM= dissTOM , cutHeigh
deepSplit=2, pamRespectsDendro = FALSE))
# Now we plot the results
plotDendroAndColors(hierTOM,
colors=data.frame(truemodule, colorStaticTOM,
colorDynamicTOM, colorDynamicHybridTOM),
dendroLabels = FALSE, marAll = c(1, 8, 3, 1),
main = "Gene dendrogram and module colors, TOM dissimilarity")
```

1.5.7 Evaluate the dissimilarity measure and branching methods used.

For each different combination we can use the Rand index as an evaluation score.

In [ ]:
```
tabStaticADJ=table(colorStaticADJ,truemodule)
tabStaticTOM=table(colorStaticTOM,truemodule)
tabDynamicADJ=table(colorDynamicADJ, truemodule)
tabDynamicTOM=table(colorDynamicTOM,truemodule)
tabDynamicHybridADJ =table(colorDynamicHybridADJ,truemodule)
tabDynamicHybridTOM =table(colorDynamicHybridTOM,truemodule)
randIndex(tabStaticADJ,adjust=F)
randIndex(tabStaticTOM,adjust=F)
randIndex(tabDynamicADJ,adjust=F)
randIndex(tabDynamicTOM,adjust=F)
randIndex(tabDynamicHybridADJ ,adjust=F)
randIndex(tabDynamicHybridTOM ,adjust=F)
```

In [ ]:
```
tabDynamicHybridTOM
```

1.6 Part 5

Network visualization.

In [ ]:
```
library(cluster)
```

1.6.1 We begin with some multi-dimensional scaling plots

In [ ]:
```
cmd1=cmdscale(as.dist(dissTOM),2)
plot(cmd1, col=as.character(colorh1), main="MDS plot",
xlab="Scaling Dimension 1", ylab="Scaling Dimension 2")
```

1.6.2 Topological overlap matrix plot for visualizing the network

In [ ]:
```
power=6
color1=colorDynamicTOM
restGenes= (color1 != "grey")
diss1=1-TOMsimilarityFromExpr( datExpr[, restGenes], power = 6 )
hier1=hclust(as.dist(diss1), method="average" )
diag(diss1) = NA;
TOMplot(diss1^4, hier1, as.character(color1[restGenes]),
main = "TOM heatmap plot, module genes" )
```

In [ ]: