Tutorial for the WGCNA package for R: III. Using simulated data to evaluate different module detection methods and gene screening approaches

3. Basic data pre-processing

Steve Horvath and Peter Langfelder

December 7, 2011

Contents

0 Setting up the R session

Before starting, the user should choose a working directory, preferably a directory devoted exclusively for this tutorial. After starting an R session, change working directory, load the requisite packages, set standard options, and load the results of previous sections:

```
# Display the current working directory
getwd();
# If necessary, change the path below to the directory where the data files are stored.
# "." means current directory. On Windows use a forward slash / instead of the usual \.
workingDir = ".";
setwd(workingDir);
# Load WGCNA package
library(WGCNA)
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
# Load the previously saved data
load("Simulated-dataSimulation.RData");
attach(ModuleEigengeneNetwork1)
```

3 Basic data pre-processing

In this section we illustrate basic data cleaning and pre-processing steps for expression data.

3.a Identification of outlying samples

We start by determining the mean expression per array and the number of missing values per array:

```
meanExpressionByArray=apply( datExpr,1,mean, na.rm=T)
NumberMissingByArray=apply( is.na(data.frame(datExpr)),1, sum)
```

A simple way to examine the mean expression per array is to use

```
sizeGrWindow(9, 5)
barplot(meanExpressionByArray,
       xlab = "Sample", ylab = "Mean expression",
       main ="Mean expression across samples",
       names.arg = c(1:50), cex.names = 0.7)
```

whose output is shown in Fig. 1. No arrays in the plot seem to have an outlying mean expression value. The numbers of missing entries in each array are

```
> NumberMissingByArray
 Sample1  Sample2  Sample3  Sample4  Sample5  Sample6  Sample7  Sample8
       0        0        0        0        0        0        0        0
 Sample9 Sample10 Sample11 Sample12 Sample13 Sample14 Sample15 Sample16
       0        0        0        0        0        0        0        0
Sample17 Sample18 Sample19 Sample20 Sample21 Sample22 Sample23 Sample24
       0        0        0        0        0        0        0        0
Sample25 Sample26 Sample27 Sample28 Sample29 Sample30 Sample31 Sample32
       0        0        0        0        0        0        0        0
Sample33 Sample34 Sample35 Sample36 Sample37 Sample38 Sample39 Sample40
       0        0        0        0        0        0        0        0
Sample41 Sample42 Sample43 Sample44 Sample45 Sample46 Sample47 Sample48
       0        0        0        0        0        0        0        0
Sample49 Sample50
       0        0
```

We note that arrays with excessive numbers of missing data should be removed, for example as

```
# Keep only arrays containing less than 500 missing entries
KeepArray= NumberMissingByArray<500
table(KeepArray)
datExpr=datExpr[KeepArray,]
y=y[KeepArray]
ArrayName[KeepArray]
```

3.b Handling missing data and zero variance in probe profiles

Here we count the number of missing samples in each probe profile, and remove probes with extensive numbers of missing samples. In addition, we remove probes that do not vary at all.

```
NumberMissingByGene =apply( is.na(data.frame(datExpr)),2, sum)
# One could do a barplot(NumberMissingByGene), but the barplot is empty in this case.
# It may be better to look at the numbers of missing samples using the summary method:
summary(NumberMissingByGene)
# Calculate the variances of the probes 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)
# Keep only genes whose variance is non-zero and have at least 4 present entries
KeepGenes= variancedatExpr>0 & no.presentdatExpr>=4
```

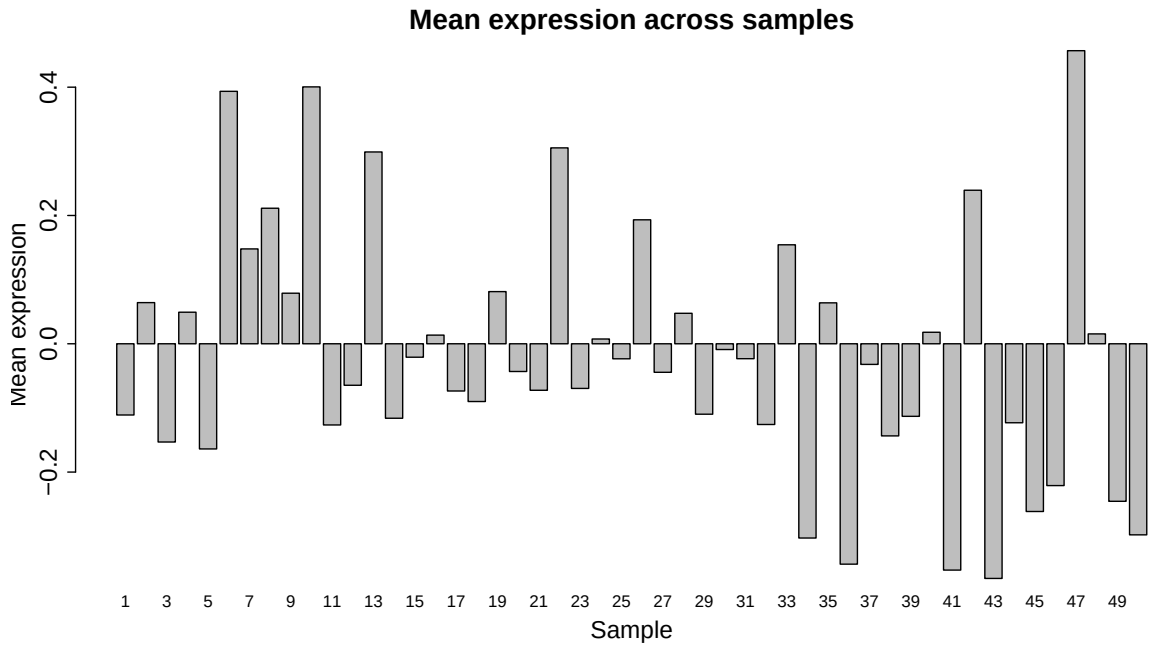

Figure 1: A barplot of mean expression (y-axis) of all probes in each sample (x-axis). No arrays in the plot seem to have an outlying mean expression value.

```
table(KeepGenes)
datExpr=datExpr[, KeepGenes]
GeneName=GeneName[KeepGenes]
```

In this case, since the data is simulated without missing data or zero-variance probes, all probes are retained.

3.c Rudimentary detection of outlier samples

We use hierarchical clustering with the Euclidean distance to determine whether there are array (sample) outliers:

```
sizeGrWindow(9, 5)
plotClusterTreeSamples(datExpr=datExpr, y=y)
```

The output is shown in Fig. 2. There are no no obvious array outliers. If there are suscpicious samples, they should be removed from the analysis. In the figure, the microarray samples are colored by the outcome y (1=black, 2=red). Since the colors dont line up with the branches, we find no evidence that samples with $y = 1$ are "globally distinct" from those with $y = 2$. When clustering microarray samples, we recommend to use the Euclidean distance. In contrast, we recommend to use the topological overlap based dissimilarity for clustering genes. Below, we investigate different methods for clustering genes (gene expression profiles).

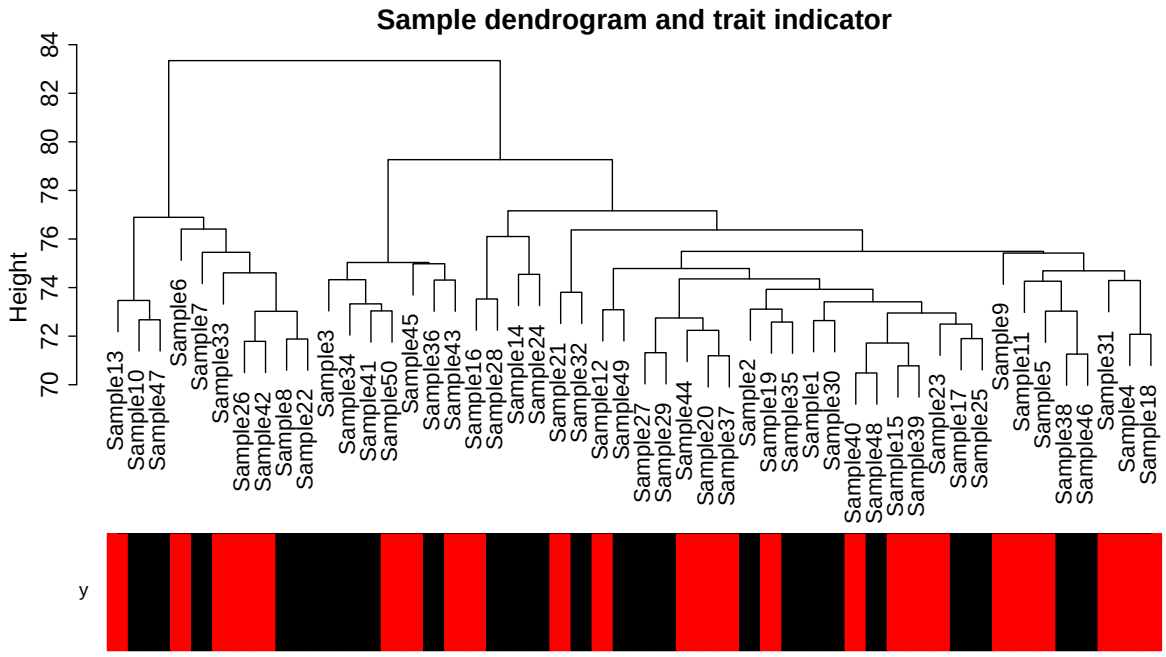

Figure 2: Clustering dendrogram of samples based on their Euclidean distance together with a color indication of the trait y (1=black, 2=red). The dendrogram shows no obvious outliers. The colors encoding the trait values do not lne up with branches, suggesting that the samples with $y = 1$ are not "globally distinct" from those with $y = 2$