Tutorial for the WGCNA package for R:
I. Network analysis of liver expression data in female mice

1. Data input and cleaning

Peter Langfelder and Steve Horvath

November 25, 2014

Contents

1 Data input, cleaning and pre-processing

This is the first step of any network analysis. We show here how to load typical expression data, pre-process them into a format suitable for network analysis, and clean the data by removing obvious outlier samples as well as genes and samples with excessive numbers of missing entries.

1.a Loading expression data

The expression data is contained in the file `LiverFemale3600.csv` that comes with this tutorial. After starting an R session, we load the requisite packages and the data, after appropriately setting the working directory:

```
# 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 the WGCNA package
library(WGCNA);
# The following setting is important, do not omit.
options(stringsAsFactors = FALSE);
#Read in the female liver data set
femData = read.csv("LiverFemale3600.csv");
# Take a quick look at what is in the data set:
dim(femData);
names(femData);
```

In addition to expression data, the data files contain extra information about the surveyed probes we do not need. One can inspect larger data frames such as `femData` by invoking R data editor via `fix(femData)`. The expression data set contains 135 samples. Note that each row corresponds to a gene and column to a sample or auxiliary information. We now remove the auxiliary data and transpose the expression data for further analysis.

```
datExpr0 = as.data.frame(t(femData[, -c(1:8)]));
names(datExpr0) = femData$substanceBXH;
rownames(datExpr0) = names(femData)[-c(1:8)];
```

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

We first check for genes and samples with too many missing values:

```
gsg = goodSamplesGenes(datExpr0, verbose = 3);
gsg$allOK
```

If the last statement returns `TRUE`, all genes have passed the cuts. If not, we remove the offending genes and samples from the data:

```
if (!gsg$allOK)
{
  # Optionally, print the gene and sample names that were removed:
  if (sum(!gsg$goodGenes)>0)
     printFlush(paste("Removing genes:", paste(names(datExpr0)[!gsg$goodGenes], collapse = ", ")));
  if (sum(!gsg$goodSamples)>0)
     printFlush(paste("Removing samples:", paste(rownames(datExpr0)[!gsg$goodSamples], collapse = ", ")));
  # Remove the offending genes and samples from the data:
  datExpr0 = datExpr0[gsg$goodSamples, gsg$goodGenes]
}
```

Next we cluster the samples (in contrast to clustering genes that will come later) to see if there are any obvious outliers.

```
sampleTree = hclust(dist(datExpr0), method = "average");
# Plot the sample tree: Open a graphic output window of size 12 by 9 inches
# The user should change the dimensions if the window is too large or too small.
sizeGrWindow(12,9)
#pdf(file = "Plots/sampleClustering.pdf", width = 12, height = 9);
par(cex = 0.6);
par(mar = c(0,4,2,0))
plot(sampleTree, main = "Sample clustering to detect outliers", sub="", xlab="", cex.lab = 1.5,
     cex.axis = 1.5, cex.main = 2)
```

It appears there is one outlier (sample `F2_221`, see Fig. 1). One can remove it by hand, or use an automatic approach. Choose a height cut that will remove the offending sample, say 15 (the red line in the plot), and use a branch cut at that height.

```
# Plot a line to show the cut
abline(h = 15, col = "red");
# Determine cluster under the line
clust = cutreeStatic(sampleTree, cutHeight = 15, minSize = 10)
table(clust)
# clust 1 contains the samples we want to keep.
keepSamples = (clust==1)
datExpr = datExpr0[keepSamples, ]
nGenes = ncol(datExpr)
nSamples = nrow(datExpr)
```

The variable `datExpr` now contains the expression data ready for network analysis.

1.c Loading clinical trait data

We now read in the trait data and match the samples for which they were measured to the expression samples.

```
traitData = read.csv("ClinicalTraits.csv");
dim(traitData)
names(traitData)

# remove columns that hold information we do not need.
allTraits = traitData[, -c(31, 16)];
allTraits = allTraits[, c(2, 11:36) ];
dim(allTraits)
names(allTraits)

# Form a data frame analogous to expression data that will hold the clinical traits.

femaleSamples = rownames(datExpr);
traitRows = match(femaleSamples, allTraits$Mice);
datTraits = allTraits[traitRows, -1];
rownames(datTraits) = allTraits[traitRows, 1];

collectGarbage();
```

We now have the expression data in the variable `datExpr`, and the corresponding clinical traits in the variable `datTraits`. Before we continue with network construction and module detection, we visualize how the clinical traits relate to the sample dendrogram.

```
# Re-cluster samples
sampleTree2 = hclust(dist(datExpr), method = "average")
# Convert traits to a color representation: white means low, red means high, grey means missing entry
traitColors = numbers2colors(datTraits, signed = FALSE);
# Plot the sample dendrogram and the colors underneath.
plotDendroAndColors(sampleTree2, traitColors,
                    groupLabels = names(datTraits),
                    main = "Sample dendrogram and trait heatmap")
```

In the plot, shown in Fig. 2, white means a low value, red a high value, and grey a missing entry.
The last step is to save the relevant expression and trait data for use in the next steps of the tutorial.

```
save(datExpr, datTraits, file = "FemaleLiver-01-dataInput.RData")
```

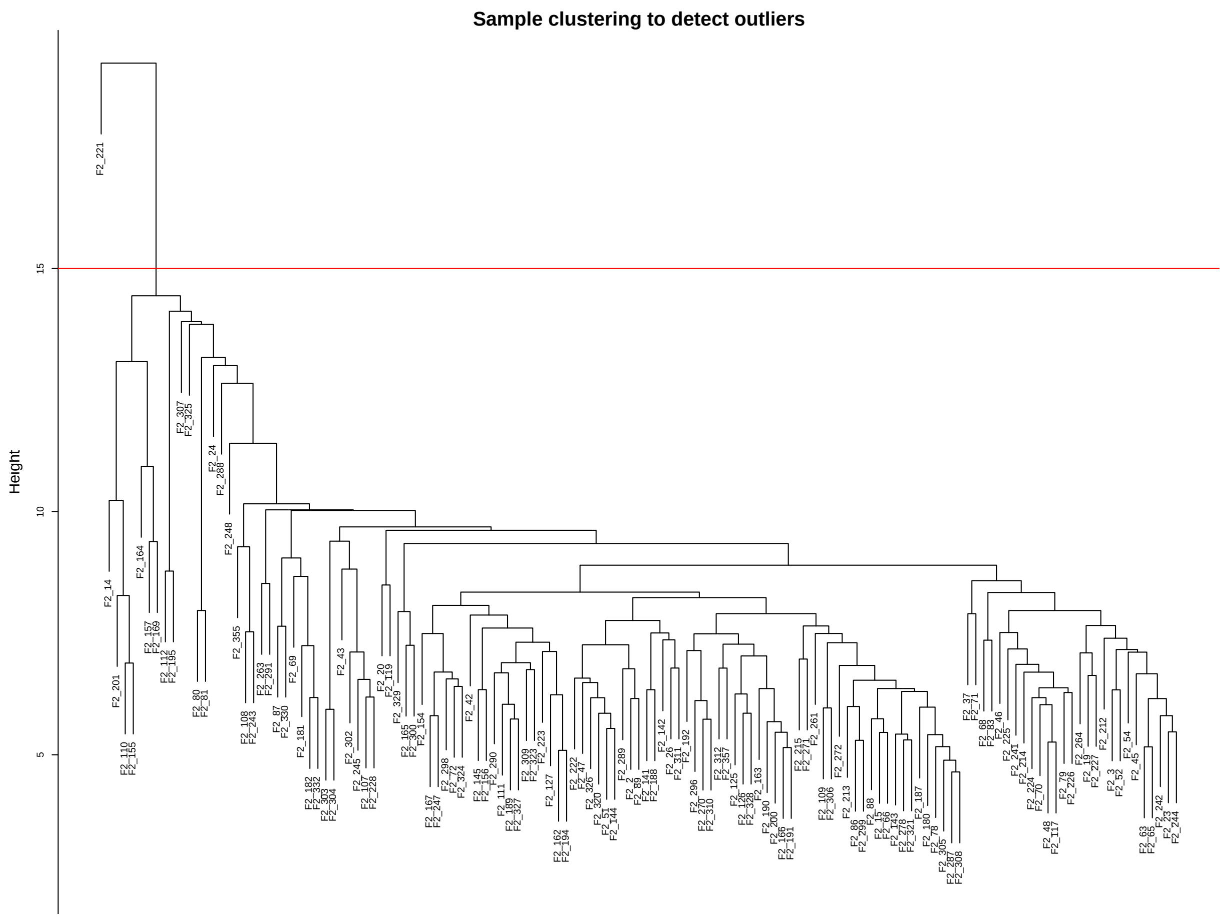

Figure 1: Clustering dendrogram of samples based on their Euclidean distance.

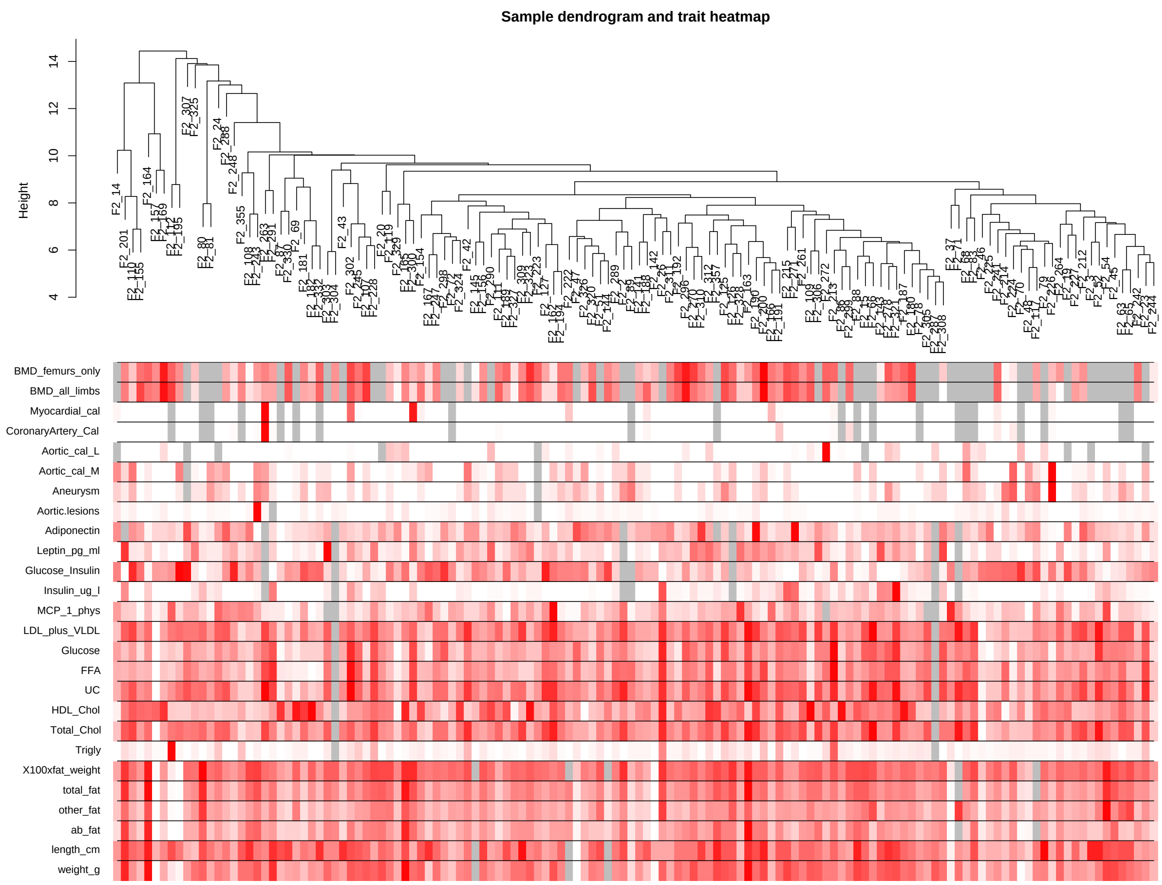

Figure 2: Clustering dendrogram of samples based on their Euclidean distance.