Network_Inference_with_WGCNA

March 31, 2021

1 WGCNA Network analysis of liver expression data in female mice

Session 6 Tutorial for Module 6 DUBII 2021

Costas Bouyioukos Universite de Paris and Anais Baudot CNRS

1.1 1. Preliminaries and data input

[ ]:
```
#  Code chunk 1
## Display the current working directory
#getwd();
## If necessary, change the path below to the directory where the data files␣
 ↪are stored.
## "." means current directory.
#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("data/LiverFemale3600.csv");
# Take a quick look at what is in the data set:
dim(femData);
names(femData);
head(femData);
```

Keep only the part of the data that contains the gene expression and keep the gene names as data frame index.

[ ]:
```
#  Code chunk 2
datExpr0 = as.data.frame(t(femData[, -c(1:8)]));
names(datExpr0) = femData$substanceBXH;
rownames(datExpr0) = names(femData)[-c(1:8)];
datExpr0
```

Check if there are genes with missing values.

[ ]:
```
#  Code chunk 3
gsg = goodSamplesGenes(datExpr0, verbose = 3);
gsg$allOK
```

All genes are OK!

Cluster the transposed matrix to identify sample outliers.

[ ]:
```
#  Code chunk 4
sampleTree = hclust(dist(datExpr0), method = "average");
# Plot the sample tree: Open a graphic output window of size 12 by 10 inches
# The user should change the dimensions if the window is too large or too small.
options(repr.plot.width = 12, repr.plot.height = 10)
plot(sampleTree, main = "Sample clustering to detect outliers", sub="", xlab="",
     cex.lab = 1.2, cex.axis = 1.5, cex.main = 2)
# Plot a line to show the cut
abline(h = 15, col = "red");
```

Identify the outlier.

[ ]:
```
#  Code chunk 5
# Determine cluster under the line
clust = cutreeStatic(sampleTree, cutHeight = 15, minSize = 10)
table(clust)
```

Remove the outlier and construct the main data frame.

[ ]:
```
#  Code chunk 5
# clust 1 contains the samples we want to keep.
keepSamples = (clust==1)
datExpr = datExpr0[keepSamples, ]
nGenes = ncol(datExpr)
nSamples = nrow(datExpr)
```

Import the clinical data, preapre and clean it.

[ ]:
```
#  Code chunk 7
traitData = read.csv("data/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];
```

Repeat the sample clustering together with a heat-map of the phenotypic data.

[ ]:
```
datTraits
```

[ ]:
```
#  Code chunk 8
# 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.
options(repr.plot.width = 15, repr.plot.height = 12)
plotDendroAndColors(sampleTree2, traitColors,
                    groupLabels = names(datTraits),
                    main = "Sample dendrogram and trait heatmap")
```

Save the analysis to an RData file.

[ ]:
```
# Code chunk 9
save(datExpr, datTraits, file = "FemaleLiver-01-dataInput.RData")
```

1.2 2. Automatic network construction and module detection

[ ]:
```
#  Code chunk 10
# Allow multi-threading within WGCNA. This helps speed up certain calculations.
# At present this call is necessary for the code to work.
# Any error here may be ignored but you may want to update WGCNA if you see one.
# See note above.
allowWGCNAThreads()
# Load the data saved in the first part
lnames = load(file = "FemaleLiver-01-dataInput.RData");
#The variable lnames contains the names of loaded variables.
lnames
```

The most convenient and automatic way to detect modules and construct a network with WGCNA. Here the developers of WGCNA are proposing a "soft thresholding" approach. This method identifies a power -to wich the correlation matrix is raised in order to calculate the network adjacency matrix- based on the criterion of scale-free approximation.

[ ]:
```
#  Code chunk 11
# Choose a set of soft-thresholding powers
powers = c(c(1:10), seq(from = 12, to = 20, by = 2))
# Call the network topology analysis function
sft = pickSoftThreshold(datExpr, powerVector = powers, verbose = 5)
```

```
# Plot the results:
par(mfrow = c(1, 2));
options(repr.plot.width = 14, repr.plot.height = 10);
# Scale-free topology fit index as a function of the soft-thresholding power
plot(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2],
     xlab = "Soft Threshold (power)", ylab = "Scale Free Topology Model␣
 ↪Fit,signed R^2", type = "n",
     main = paste("Scale independence"));
text(sft$fitIndices[,1], -sign(sft$fitIndices[,3])*sft$fitIndices[,2],
     labels = powers, cex = 0.9, col = "red");
# this line corresponds to using an R^2 cut-off of h
abline(h = 0.90, col = "red")
# Mean connectivity as a function of the soft-thresholding power
plot(sft$fitIndices[,1], sft$fitIndices[,5],
     xlab = "Soft Threshold (power)", ylab = "Mean Connectivity", type = "n",
     main = paste("Mean connectivity"))
text(sft$fitIndices[,1], sft$fitIndices[,5], labels = powers, cex = 0.9, col =␣
 ↪"red")
```

1.2.1 The actual network construction step.

We choose 6 (or 7 for signed) as the lowest power that constructs a scale free topology. And then we instruct the function to generate modules of size 30, merge modules which are more than 25% similar and save the Topological Overlap Matrix in an object.

[ ]:
```
#  Code chunk 12
net = blockwiseModules(datExpr, power = 7,
                       TOMType = "signed", minModuleSize = 30,
                       reassignThreshold = 1e-6, mergeCutHeight = 0.25,
                       numericLabels = TRUE, pamRespectsDendro = FALSE,
                       saveTOMs = TRUE, nThreads = 8,
                       saveTOMFileBase = "femaleMouseTOM",
                       verbose = 3)
```

Here is the modules (as numbers and not colours yet) of each module with its size.

[ ]:
```
colnames(net$MEs)
```

Here is the resuling plot dendrogram of the module construction and the clustering of the genes.

[ ]:
```
#  Code chunk 13
# Convert labels to colors for plotting
mergedColors = labels2colors(net$colors)
#mergedColors
# Plot the dendrogram and the module colors underneath
plotDendroAndColors(net$dendrograms[[1]], mergedColors[net$blockGenes[[1]]],
                    "Module colors",
```

```
                    dendroLabels = FALSE, hang = 0.03,
                    addGuide = TRUE, guideHang = 0.05)
```

Save results of this part in an .RData file.

[ ]:
```
#  Code chunk 14
moduleLabels = net$colors
moduleColors = labels2colors(net$colors)
MEs = net$MEs;
geneTree = net$dendrograms[[1]];
save(MEs, moduleLabels, moduleColors, geneTree,
     file = "FemaleLiver-02-networkConstruction-auto.RData")
```

1.3 3. Relating modules to external information and identifying important genes

[ ]:
```
#  Code chunk 15
lnames = load(file = "FemaleLiver-01-dataInput.RData");
#The variable lnames contains the names of loaded variables.
lnames
# Load network data saved in the second part.
lnames = load(file = "FemaleLiver-02-networkConstruction-auto.RData");
lnames
```

1.3.1 Quantifying module–trait associations

Here we identify modules that are significantly associated with the measured clinical traits. We already have a computed summary profile (eigengene) for each module, so then we simply correlate eigengenes with phenotypic traits and look for the most significant associations:

[ ]:
```
#  Code chunk 16
# Define numbers of genes and samples
nGenes = ncol(datExpr);
nSamples = nrow(datExpr);
# Recalculate MEs with color labels
MEs0 = moduleEigengenes(datExpr, moduleColors)$eigengenes
MEs = orderMEs(MEs0)
moduleTraitCor = cor(MEs, datTraits, use = "p");
moduleTraitPvalue = corPvalueStudent(moduleTraitCor, nSamples);
```

Visualise the module-trait association. Each module eigengene and its correlation coefficient are ploted here. Since we have many a colour code aids the interprettion of the plot.

[ ]:
```
#  Code chunk 17
options(repr.plot.width=16, repr.plot.height=12)
# Will display correlations and their p-values
textMatrix =  paste(signif(moduleTraitCor, 2), "\n(",
```

```
                           signif(moduleTraitPvalue, 1), ")", sep = "");
dim(textMatrix) = dim(moduleTraitCor)
par(mar = c(6, 11, 1, 0));
# Display the correlation values within a heatmap plot
labeledHeatmap(Matrix = moduleTraitCor,
               xLabels = names(datTraits),
               yLabels = names(MEs),
               ySymbols = names(MEs),
               colorLabels = FALSE,
               colors = blueWhiteRed(50),
               textMatrix = textMatrix,
               setStdMargins = FALSE,
               cex.text = 0.5,
               zlim = c(-1,1),
               main = paste("Module-trait relationships"))
```

[ ]:
```
#  Code chunk 18
names(datExpr)[moduleColors=="salmon"]
```

Probe annotation file provided by the manufacturer to facilitate functional annotation.

[ ]:
```
#  Code chunk 19
annot = read.csv(file = "data/GeneAnnotation.csv");
dim(annot)
names(annot)
probes = names(datExpr)
probes2annot = match(probes, annot$substanceBXH)
# The following is the number or probes without annotation:
sum(is.na(probes2annot))
# Should return 0.
```

Collect all the information for significant genes related to body weight.

[ ]:
```
weight = as.data.frame(datTraits$weight_g);
geneModuleMembership = as.data.frame(cor(datExpr, MEs, use = "p"));
MMPvalue = as.data.frame(corPvalueStudent(as.matrix(geneModuleMembership),␣
 ↪nSamples));
modNames = substring(names(MEs), 3);
names(geneModuleMembership) = paste("MM", modNames, sep="");
names(MMPvalue) = paste("p.MM", modNames, sep="");
geneTraitSignificance = as.data.frame(cor(datExpr, weight, use = "p"));
GSPvalue = as.data.frame(corPvalueStudent(as.matrix(geneTraitSignificance),␣
 ↪nSamples));
names(geneTraitSignificance) = paste("GS.", names(weight), sep="");
names(GSPvalue) = paste("p.GS.", names(weight), sep="");

#  Code chunk 20
```

```
# Create the starting data frame
geneInfo0 = data.frame(substanceBXH = probes,
                       geneSymbol = annot$gene_symbol[probes2annot],
                       LocusLinkID = annot$LocusLinkID[probes2annot],
                       moduleColor = moduleColors,
                       geneTraitSignificance,
                       GSPvalue)
# Order modules by their significance for weight
modOrder = order(-abs(cor(MEs, weight, use = "p")));
# Add module membership information in the chosen order
for (mod in 1:ncol(geneModuleMembership))
{
  oldNames = names(geneInfo0)
  geneInfo0 = data.frame(geneInfo0, geneModuleMembership[, modOrder[mod]],
                         MMPvalue[, modOrder[mod]]);
  names(geneInfo0) = c(oldNames, paste("MM.", modNames[modOrder[mod]], sep=""),
                       paste("p.MM.", modNames[modOrder[mod]], sep=""))
}

# Order the genes in the geneInfo variable first by module color, then by␣
 ↪geneTraitSignificance
geneOrder = order(geneInfo0$moduleColor, -abs(geneInfo0$GS.datTraits.weight_g));
geneInfo = geneInfo0[geneOrder, ]
```

Save the results in an output file for further analysis.

[ ]:
```
#  Code chunk 21
write.csv(geneInfo, file = "geneInfo.csv")
```

1.4 4. Interfacing network analysis with other data such as functional annotation and gene ontology

[ ]:
```
#  Code chunk 22
# Load the expression and trait data saved in the first part
lnames = load(file = "FemaleLiver-01-dataInput.RData");
#The variable lnames contains the names of loaded variables.
lnames
# Load network data saved in the second part.
lnames = load(file = "FemaleLiver-02-networkConstruction-auto.RData");
lnames
```

[ ]:
```
#  Code chunk 23
# Read in the probe annotation
annot = read.csv(file = "data/GeneAnnotation.csv");
# Match probes in the data set to the probe IDs in the annotation file
probes = names(datExpr)
probes2annot = match(probes, annot$substanceBXH)
```

```
# Get the corresponding Locuis Link IDs
allLLIDs = annot$LocusLinkID[probes2annot];
# $ Choose interesting modules
intModules = c("brown", "red", "salmon")
for (module in intModules)
{
  # Select module probes
  modGenes = (moduleColors==module)
  # Get their entrez ID codes
  modLLIDs = allLLIDs[modGenes];
  # Write them into a file
  fileName = paste("LocusLinkIDs-", module, ".txt", sep="");
  write.table(as.data.frame(modLLIDs), file = fileName,
              row.names = FALSE, col.names = FALSE)
}
# As background in the enrichment analysis, we will use all probes in the␣
 ↪analysis.
fileName = paste("LocusLinkIDs-all.txt", sep="");
write.table(as.data.frame(allLLIDs), file = fileName,
            row.names = FALSE, col.names = FALSE)
```

[ ]:
```
#  Code chunk 24
GOenr = GOenrichmentAnalysis(moduleColors, allLLIDs, organism = "mouse", nBestP␣
 ↪= 10);
```

[ ]:
```
#anRichment(moduleColors, allLLIDs, organism = "mouse", nBestP = 10);  # Does␣
 ↪not work yet.
```

[ ]:
```
#  Code chunk 25
tab = GOenr$bestPTerms[[4]]$enrichment
```

[ ]:
```
#  Code chunk 26
names(tab)
```

[ ]:
```
#  Code chunk 27
write.table(tab, file = "GOEnrichmentTable.csv", sep = ",", quote = TRUE, row.
 ↪names = FALSE)
```

[ ]:
```
#  Code chunk 28
keepCols = c(1, 2, 5, 6, 7, 12, 13);
screenTab = tab[, keepCols];
# Round the numeric columns to 2 decimal places:
numCols = c(3, 4);
screenTab[, numCols] = signif(apply(screenTab[, numCols], 2, as.numeric), 2)
# Truncate the the term name to at most 40 characters
screenTab[, 7] = substring(screenTab[, 7], 1, 40)
# Shorten the column names:
```

```
colnames(screenTab) = c("module", "size", "p-val", "Bonf", "nInTerm", "ont",␣
 ↪"term name");
rownames(screenTab) = NULL;
# Set the width of R's output. The reader should play with this number to␣
 ↪obtain satisfactory output.
options(width=95)
# Finally, display the enrichment table:
screenTab
```

1.5 5. Export of networks to external software

[ ]:
```
#  Code chunk 29
# Load the expression and trait data saved in the first part
lnames = load(file = "FemaleLiver-01-dataInput.RData");
#The variable lnames contains the names of loaded variables.
lnames
# Load network data saved in the second part.
lnames = load(file = "FemaleLiver-02-networkConstruction-auto.RData");
lnames
```

[ ]:
```
#  Code chunk 30
# Recalculate topological overlap if needed
TOM = TOMsimilarityFromExpr(datExpr, power = 7);
# Read in the annotation file
annot = read.csv(file = "data/GeneAnnotation.csv");
# Select modules
modules = c("brown", "blue");
# Select module probes
probes = names(datExpr)
inModule = is.finite(match(moduleColors, modules));
modProbes = probes[inModule];
modGenes = annot$gene_symbol[match(modProbes, annot$substanceBXH)];
# Select the corresponding Topological Overlap
modTOM = TOM[inModule, inModule];
dimnames(modTOM) = list(modProbes, modProbes)
# Export the network into edge and node list files Cytoscape can read
cyt = exportNetworkToCytoscape(modTOM,
  edgeFile = paste("CytoscapeInput-edges-", paste(modules, collapse="-"), ".
 ↪txt", sep=""),
  nodeFile = paste("CytoscapeInput-nodes-", paste(modules, collapse="-"), ".
 ↪txt", sep=""),
  weighted = TRUE,
  threshold = 0.1,
  nodeNames = modProbes,
  altNodeNames = modGenes,
  nodeAttr = moduleColors[inModule]);
```

Open these two files as node table and edge table with Cytoscape and inspect the network of the brown-red modules

[ ]: