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

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

Peter Langfelder and Steve Horvath

November 25, 2014

Contents

0 Preliminaries: setting up the R session

Here we assume that a new R session has just been started. We load the WGCNA package, set up basic parameters and load data saved in previous parts of the tutorial.

```
# 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);
# 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
```

We use the network file obtained by the step-by-step network construction and module detection; we encourage the reader to use the results of the other approaches as well.

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

Our previous analysis has identified several modules (labeled brown, red, and salmon) that are highly associated with weight. To facilitate a biological interpretation, we would like to know the gene ontologies of the genes in the modules, whether they are significantly enriched in certain functional categories etc.

4.a Output gene lists for use with online software and services

One option is to simply export a list of gene identifiers that can be used as input for several popular gene ontology and functional enrichment analysis suites such as DAVID or AmiGO. For example, we write out the LocusLinkID (entrez) codes for the brown module into a file:

```
# Read in the probe annotation
annot = read.csv(file = "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)
```

4.b Enrichment analysis directly within R

The WGCNA package now contains a function to perform GO enrichment analysis using a simple, single step. To run the function, Biconductor packages GO.db, AnnotationDBI, and the appropriate organism-specific annotation package(s) need to be installed before running this code. The organism-specific packages have names of the form org.Xx.eg.db, where Xx stands for organism code, for example, Mm for mouse, Hs for human, etc. The only exception is yeast, for which no org.Xx.eg.db package is available; instead, the package carries the name org.Sc.sgd.db. Please visit the Bioconductor main page at http://www.bioconductor.org to download and install the required packages. In our case we are studying gene expressions from mice, so this code needs the package org.Mm.eg.db. Calling the GO enrichment analysis function `GOenrichmentAnalysis` is very simple. The function takes a vector of module labels, and the Entrez (a.k.a. Locus Link) codes for the genes whose labels are given.

```
GOenr = GOenrichmentAnalysis(moduleColors, allLLIDs, organism = "mouse", nBestP = 10);
```

The function runs for awhile and returns a long list, the most interesting component of which is

```
tab = GOenr$bestPTerms[[4]]$enrichment
```

This is an enrichment table containing the 10 best terms for each module present in `moduleColors`. Names of the columns within the table can be accessed by

```
names(tab)
```

We refer the reader to the help page of the function within R (available using `?GOenrichmentAnalysis` at the R prompt) for details of what each column means. Because the term definitions can be quite long, the table is a bit difficult to display on the screen. For readers who prefer to look at tables in Excel or similar spreadsheet software, it is best to save the table into a file and open it using their favorite tool:

```
write.table(tab, file = "GOEnrichmentTable.csv", sep = ",", quote = TRUE, row.names = FALSE)
```

On the other hand, to quickly take a look at the results, one can also abridge the table a bit and display it directly on screen:

```
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
```

The table is quite long and the reader will have to scroll through the output to see it all. Several of the modules have very significant enrichment, for example the blue, brown, and salmon modules.