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

1. Simulation of expression and trait data

Steve Horvath and Peter Langfelder

December 7, 2011

Contents

0 Setting up the R session **1**

1 Simulation of expression and trait data **1**
1.a Building the module structure 1
1.b Simulating gene expressions around the module eigengenes 2

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 and set standard options:

```
# 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);
```

1 Simulation of expression and trait data

In this section we illustrate simulation of expression data with a simple module structure.

1.a Building the module structure

We choose the basic parameters of the simulated data set: we will simulate 3000 genes in 50 samples. The genes will fall into five proper modules (labeled turquoise, blue, brown, green, and yellow) and a relatively large number of genes will be simulated outside of the proper modules ("grey" genes).

```
# Here are input parameters of the simulation model
# number of samples or microarrays in the training data
no.obs=50
# now we specify the true measures of eigengene significance
# recall that ESturquoise=cor(y,MEturquoise)
ESturquoise=0; ESbrown= -.6;
ESgreen=.6;ESyellow=0
# Note that we dont specify the eigengene significance of the blue module
# since it is highly correlated with the turquoise module.
ESvector=c(ESturquoise,ESbrown,ESgreen,ESyellow)
# number of genes
nGenes1=3000
# proportion of genes in the turquoise, blue, brown, green, and yellow module #respectively.
simulateProportions1=c(0.2,0.15, 0.08, 0.06, 0.04)
# Note that the proportions dont add up to 1. The remaining genes will be colored grey,
# ie the grey genes are non-module genes.
# set the seed of the random number generator. As a homework exercise change this seed.
set.seed(1)
#Step 1: simulate a module eigengene network.
# Training Data Set I
MEgreen=rnorm(no.obs)
scaledy=MEgreen*ESgreen+sqrt(1-ESgreen^2)*rnorm(no.obs)
y=ifelse( scaledy>median(scaledy),2,1)
MEturquoise= ESturquoise*scaledy+sqrt(1-ESturquoise^2)*rnorm(no.obs)
# we simulate a strong dependence between MEblue and MEturquoise
MEblue= .6*MEturquoise+ sqrt(1-.6^2) *rnorm(no.obs)
MEbrown= ESbrown*scaledy+sqrt(1-ESbrown^2)*rnorm(no.obs)
MEyellow= ESyellow*scaledy+sqrt(1-ESyellow^2)*rnorm(no.obs)
ModuleEigengeneNetwork1=data.frame(y,MEturquoise,MEblue,MEbrown,MEgreen, MEyellow)
```

The variable `ModuleEigengeneNetwork1` contains the "seed" eigengenes and a simulated clinical trait `y`. The eigengene network can be simply inspected by

```
> signif(cor(ModuleEigengeneNetwork1, use="p"),2)
                y MEturquoise MEblue MEbrown MEgreen MEyellow
y            1.00       0.050 -0.110  -0.260   0.450     0.11
MEturquoise  0.05       1.000  0.670   0.100   0.026     0.12
MEblue      -0.11       0.670  1.000   0.066  -0.190    -0.12
MEbrown     -0.26       0.100  0.066   1.000  -0.270     0.11
MEgreen      0.45       0.026 -0.190  -0.270   1.000    -0.12
MEyellow     0.11       0.120 -0.120   0.110  -0.120     1.00
```

1.b Simulating gene expressions around the module eigengenes

The package contains a convenient function to simulate five modules which we call below

```
dat1=simulateDatExpr5Modules(MEturquoise=ModuleEigengeneNetwork1$MEturquoise,
        MEblue=ModuleEigengeneNetwork1$MEblue,
        MEbrown=ModuleEigengeneNetwork1$MEbrown,
        MEyellow=ModuleEigengeneNetwork1$MEyellow,
        MEgreen=ModuleEigengeneNetwork1$MEgreen,
        nGenes=nGenes1,
        simulateProportions=simulateProportions1)
```

The simulated data (dat1) is a list with the following components

```
> names(dat1)
[1] "datExpr"    "truemodule" "datME"
```

We attach the data "into the main search path" so we can use the component names directly, without referring to `dat1`:

```
datExpr = dat1$datExpr;
truemodule = dat1$truemodule;
datME = dat1$datME;
attach(ModuleEigengeneNetwork1)
```

To see what is in the data, we can simply type

```
table(truemodule)
dim(datExpr)
```

with the result

```
> dim(datExpr)
[1]   50 3000
> table(truemodule)
truemodule
     blue     brown     green      grey turquoise    yellow
      450       240       120      1410       600       180
```

The output indicated we simulated 3000 genes in 50 samples, and a large number of the genes are "grey", i.e., genes that are not part of any proper module. the next piece of code assigns gene and sample names to columns and rows of the expression data.

```
datExpr=data.frame(datExpr)
ArrayName=paste("Sample",1:dim(datExpr)[[1]], sep="" )
# The following code is useful for outputting the simulated data
GeneName=paste("Gene",1:dim(datExpr)[[2]], sep="" )
dimnames(datExpr)[[1]]=ArrayName
dimnames(datExpr)[[2]]=GeneName
```

We now remove data we will not need and save the results of this session for use in subsequent sections.

```
rm(dat1); collectGarbage();
# The following command will save all variables defined in the current session.
save.image("Simulated-dataSimulation.RData");
```