<https://github.com/Prabhashkjha/CAD_lncRNA_WGCNA>

##Loading expression data

#Display the current working directory

getwd();

"C:/Users/Prabhash/Documents"

# 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 = "C:/Users/Prabhash/Desktop";

setwd(workingDir);

# Load the WGCNA package

library(WGCNA);

# The following setting is important, do not omit.

options(stringsAsFactors = FALSE);

#Read in the data set

femData = read.csv("lm.csv");

# Take a quick look at what is in the data set:

dim(femData);

names(femData);

datExpr0 = as.data.frame(t(femData[, -c(1:8)]));

names(datExpr0) = femData$substanceBXH;

rownames(datExpr0) = names(femData)[-c(1:8)];

## Checking data for excessive missing values and identication 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 o  
ending 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)

# Plot a line to show the cut

abline(h = 40, 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)

save(datExpr, file = "dataInput.RData")

enableWGCNAThreads()

# Load the data saved in the first part

lnames = load(file = "dataInput.RData");

#The variable lnames contains the names of loaded variables.

lnames

## Step by step network construction and module detection

# Choosing the soft-thresholding power: analysis of network topology

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:

sizeGrWindow(9, 5)

par(mfrow = c(1,2));

cex1 = 0.9;

# 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=cex1,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=cex1,col="red")

# Co-expression similarity and adjacency

softPower = 6;

adjacency = adjacency(datExpr, power = softPower);

# Turn adjacency into topological overlap

TOM = TOMsimilarity(adjacency);

dissTOM = 1-TOM

# Clustering using TOM

# Call the hierarchical clustering function

geneTree = hclust(as.dist(dissTOM), method = "average");

# Plot the resulting clustering tree (dendrogram)

sizeGrWindow(12,9)

plot(geneTree, xlab="", sub="", main = "Gene clustering on TOM-based dissimilarity",

labels = FALSE, hang = 0.04);

# We like large modules, so we set the minimum module size relatively high:

minModuleSize = 30;

# Module identification using dynamic tree cut:

dynamicMods = cutreeDynamic(dendro = geneTree, distM = dissTOM,

deepSplit = 2, pamRespectsDendro = FALSE,

minClusterSize = minModuleSize);

table(dynamicMods)

# Convert numeric lables into colors

dynamicColors = labels2colors(dynamicMods)

table(dynamicColors)

# Plot the dendrogram and colors underneath

sizeGrWindow(8,6)

plotDendroAndColors(geneTree, dynamicColors, "Dynamic Tree Cut",

dendroLabels = FALSE, hang = 0.03,

addGuide = TRUE, guideHang = 0.05,

main = "Gene dendrogram and module colors")

# Merging of modules whose expression proles are very similar

# Calculate eigengenes

MEList = moduleEigengenes(datExpr, colors = dynamicColors)

MEs = MEList$eigengenes

# Calculate dissimilarity of module eigengenes

MEDiss = 1-cor(MEs);

# Cluster module eigengenes

METree = hclust(as.dist(MEDiss), method = "average");

# Plot the result

sizeGrWindow(7, 6)

plot(METree, main = "Clustering of module eigengenes",

xlab = "", sub = "")

MEDissThres = 0.25

# Plot the cut line into the dendrogram

abline(h=MEDissThres, col = "red")

# Call an automatic merging function

merge = mergeCloseModules(datExpr, dynamicColors, cutHeight = MEDissThres, verbose = 3)

# The merged module colors

mergedColors = merge$colors;

# Eigengenes of the new merged modules:

mergedMEs = merge$newMEs;

sizeGrWindow(12, 9)

#pdf(file = "Plots/geneDendro-3.pdf", wi = 9, he = 6)

plotDendroAndColors(geneTree, cbind(dynamicColors, mergedColors),

c("Dynamic Tree Cut", "Merged dynamic"),

dendroLabels = FALSE, hang = 0.03,

addGuide = TRUE, guideHang = 0.05)

# Rename to moduleColors

moduleColors = mergedColors

# Construct numerical labels corresponding to the colors

colorOrder = c("grey", standardColors(50));

moduleLabels = match(moduleColors, colorOrder)-1;

MEs = mergedMEs;

# Save module colors and labels for use in subsequent parts

save(MEs, moduleLabels, moduleColors, geneTree, file = "networkConstruction.RData")

# Load the expression and trait data saved in the first part

lnames = load(file = "dataInput.RData");

#The variable lnames contains the names of loaded variables.

lnames

# Load network data saved in the second part.

lnames = load(file = "networkConstruction.RData");

lnames

nGenes = ncol(datExpr)

nSamples = nrow(datExpr)

## Visualizing the gene network

# Calculate topological overlap anew: this could be done more efficiently by saving the TOM

# calculated during module detection, but let us do it again here.

dissTOM = 1-TOMsimilarityFromExpr(datExpr, power = 6);

# Transform dissTOM with a power to make moderately strong connections more visible in the heatmap

plotTOM = dissTOM^7;

# Set diagonal to NA for a nicer plot

diag(plotTOM) = NA;

# Call the plot function

sizeGrWindow(9,9)

TOMplot(plotTOM, geneTree, moduleColors, main = "Network heatmap plot, all genes")

## Exporting network data to network visualization software

# Recalculate topological overlap

TOM = TOMsimilarityFromExpr(datExpr, power = 6);

# Read in the annotation file

annot = read.csv(file = "GeneAnnotation.csv");

# Select module--- top modules

module = "brown";

# Select module probes

probes = names(datExpr)

inModule = (moduleColors==module);

modProbes = probes[inModule];

# Select the corresponding Topological Overlap

modTOM = TOM[inModule, inModule];

dimnames(modTOM) = list(modProbes, modProbes)

# Export the network into an edge list file VisANT can read

vis = exportNetworkToVisANT(modTOM,

file = paste("VisANTInput-", module, ".txt", sep=""),

weighted = TRUE,

threshold = 0,

probeToGene = data.frame(annot$substanceBXH, annot$gene\_symbol) )

# Restrict the gene output in larger modules

nTop = 30;

IMConn = softConnectivity(datExpr[, modProbes]);

top = (rank(-IMConn) <= nTop)

vis = exportNetworkToVisANT(modTOM[top, top],

file = paste("VisANTInput-", module, "-top30.txt", sep=""),

weighted = TRUE,

threshold = 0,

probeToGene = data.frame(annot$substanceBXH, annot$gene\_symbol) )