Corrected R code from chapter 12 of the book

Horvath S (2011) Weighted Network Analysis. Applications in Genomics and Systems Biology. Springer Book. ISBN: 978-1-4419-8818-8

Steve Horvath (Email: shorvath At mednet.ucla.edu)

Introduction

Integrated weighted correlation network analysis of mouse liver gene expression data Chapter 12 and this R software tutorial describe a case study for carrying out an integrated weighted correlation network analysis of mouse gene expression, sample trait, and genetic marker data. It describes how to i) use sample networks (signed correlation networks) for detecting outlying observations, ii) find co-expression modules and key genes related to mouse body weight and other physiologic traits in female mice, iii) study module preservation between female and male mice, iv) carry out a systems genetic analysis with the network edge orienting approach to find causal genes for body weight, v) define consensus modules between female and male mice. We also describe methods and software for visualizing networks and for carrying out gene ontology enrichment analysis.

The mouse cross is described in section 5.5 (and reference Ghazalpour et al 2006). The mouse gene expression data were generated by the labs of Jake Lusis and Eric Schadt. Here we used a mouse liver gene expression data set which contains 3600 gene expression profiles. These were filtered from the original over 20,000 genes by keeping only the most variant and most connected probes. In addition to the expression data, several physiological quantitative traits were measured for the mice, e.g. body weight. The expression data is contained in the file "LiverFemale3600.csv" that can be found at the following webpages:

http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/Book/MouseData.zip
http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/Book/GeneAnnotation.zip

or

http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/Rpackages/WGCNA/MouseData.zip
http://www.genetics.ucla.edu/labs/horvath/CoexpressionNetwork/Rpackages/WGCNA/GeneAnnotation.zip

Set up local data directory (used below).

```
data.dir <- "MouseData"
```

Much of the following R code was created by Peter Langfelder.

Important comment: This material is slightly different from what is presented in the book. There were small errors in the R code of the book chapter which have been corrected.

Section 12.1 Constructing a sample network for outlier detection

Here we use sample network methods for finding outlying microarray samples (see section 7.7). Specifically, the Euclidean distance based sample network is simply the canonical Euclidean distance based network `A(uv)=1-||S(u)-S(v)||^2/maxDiss` (Eq 7.18). Next we use the standardized connectivity `Z.ku=(ku-mean(k))/(sqrt(var(k)))` (Eq. 7.24) to identify array outliers. In the following, we present relevant R code.

```
suppressMessages(library(WGCNA))
```

```
## ==========================================================================
## *
## *  Package WGCNA 1.27.1 loaded.
## *
## ==========================================================================
```

```
allowWGCNAThreads()
```

```
## Allowing multi-threading with up to 4 threads.
```

```
suppressMessages(library(cluster))
options(stringsAsFactors = FALSE)

# Read in the female liver data set
femData = read.csv(file.path(data.dir, "LiverFemale3600.csv"))
dim(femData)
```

```
## [1] 3600  143
```

Note there are 3600 genes and 143 columns. The column names are

```
# the first 8 columns contain gene information
names(femData)[1:9]
```

```
## [1] "substanceBXH"   "gene_symbol"    "LocusLinkID"    "ProteomeID"
## [5] "cytogeneticLoc" "CHROMOSOME"     "StartPosition"  "EndPosition"
## [9] "F2_2"
```

```
# Remove gene information and transpose the expression data
datExprFemale = as.data.frame(t(femData[, -c(1:8)]))
names(datExprFemale) = femData$substanceBXH
rownames(datExprFemale) = names(femData)[-c(1:8)]
```

```
# Now we read in the physiological trait data
traitData = read.csv(file.path(data.dir, "ClinicalTraits.csv"))
dim(traitData)
```

```
## [1] 361  38
```

```
names(traitData)
```

```
##  [1] "X"                 "Mice"              "Number"
##  [4] "Mouse_ID"          "Strain"            "sex"
##  [7] "DOB"               "parents"           "Western_Diet"
## [10] "Sac_Date"          "weight_g"          "length_cm"
## [13] "ab_fat"            "other_fat"         "total_fat"
## [16] "comments"          "X100xfat_weight"   "Trigly"
## [19] "Total_Chol"        "HDL_Chol"          "UC"
## [22] "FFA"               "Glucose"           "LDL_plus_VLDL"
## [25] "MCP_1_phys"        "Insulin_ug_l"      "Glucose_Insulin"
## [28] "Leptin_pg_ml"      "Adiponectin"       "Aortic.lesions"
## [31] "Note"              "Aneurysm"          "Aortic_cal_M"
## [34] "Aortic_cal_L"      "CoronaryArtery_Cal" "Myocardial_cal"
## [37] "BMD_all_limbs"     "BMD_femurs_only"
```

```
# use only a subset of the columns
allTraits = traitData[, c(2, 11:15, 17:30, 32:38)]
names(allTraits)
```

```
##  [1] "Mice"              "weight_g"          "length_cm"
##  [4] "ab_fat"            "other_fat"         "total_fat"
##  [7] "X100xfat_weight"   "Trigly"            "Total_Chol"
## [10] "HDL_Chol"          "UC"                "FFA"
## [13] "Glucose"           "LDL_plus_VLDL"     "MCP_1_phys"
## [16] "Insulin_ug_l"      "Glucose_Insulin"   "Leptin_pg_ml"
## [19] "Adiponectin"       "Aortic.lesions"    "Aneurysm"
## [22] "Aortic_cal_M"      "Aortic_cal_L"      "CoronaryArtery_Cal"
## [25] "Myocardial_cal"    "BMD_all_limbs"     "BMD_femurs_only"
```

```
# Order the rows of allTraits so that they match those of datExprFemale:
Mice = rownames(datExprFemale)
traitRows = match(Mice, allTraits$Mice)
datTraits = allTraits[traitRows, -1]
rownames(datTraits) = allTraits[traitRows, 1]
# show that row names agree
table(rownames(datTraits) == rownames(datExprFemale))
```

```
##
## TRUE
##  135
```

Message: the traits and expression data have been aligned correctly.

```
# sample network based on squared Euclidean distance note that we
# transpose the data
A = adjacency(t(datExprFemale), type = "distance")
# this calculates the whole network connectivity
k = as.numeric(apply(A, 2, sum)) - 1
# standardized connectivity
Z.k = scale(k)


# Designate samples as outlying if their Z.k value is below the threshold
thresholdZ.k = -5  # often -2.5

# the color vector indicates outlyingness (red)
outlierColor = ifelse(Z.k < thresholdZ.k, "red", "black")

# calculate the cluster tree using flahsClust or hclust
sampleTree = flashClust(as.dist(1 - A), method = "average")
# Convert traits to a color representation: where red indicates high
# values
traitColors = data.frame(numbers2colors(datTraits, signed = FALSE))
dimnames(traitColors)[[2]] = paste(names(datTraits), "C", sep = "")
datColors = data.frame(outlierC = outlierColor, traitColors)
# Plot the sample dendrogram and the colors underneath.
plotDendroAndColors(sampleTree, groupLabels = names(datColors), colors = datColors,
    main = "Sample dendrogram and trait heatmap")
```

Caption: Cluster tree of mouse liver samples. The leaves of the tree correspond to the mice. The first color band underneath the tree indicates which arrays appear to be outlying. The second color band represents body weight (red indicates high value). Similarly, the remaining color-bands color-code the numeric values of physiologic traits.

The Figure shows the resulting cluster tree where each leaf corresponds to a mouse sample. The first color-band underneath the tree indicates which samples appear outlying (colored in red) according to a low value. The mouse labelled `F2_221` is highly outlying `(Z.k<-5)`, which is why we remove it from the subsequent analysis. The other color bands color-code physiological traits. Note that the outlying samples do not appear to have systematically different physiologic trait values. Although the outlying samples are suspicious, we will keep them in the subsequent analysis.

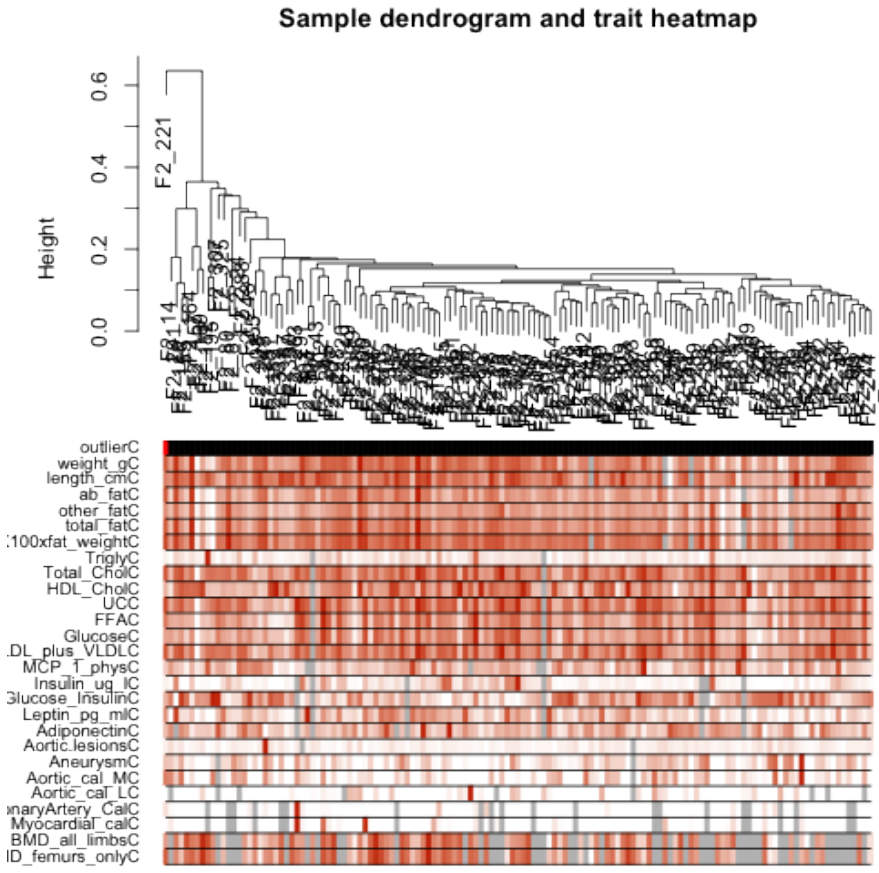

Figure 1: plot of chunk unnamed-chunk-8

```
# Remove outlying samples from expression and trait data
remove.samples = Z.k < thresholdZ.k | is.na(Z.k)
# the following 2 lines differ from what is written in the book
datExprFemale = datExprFemale[!remove.samples, ]
datTraits = datTraits[!remove.samples, ]
# Recompute the sample network among the remaining samples
A = adjacency(t(datExprFemale), type = "distance")
# Let's recompute the Z.k values of outlyingness
k = as.numeric(apply(A, 2, sum)) - 1
Z.k = scale(k)
```

Section 12.2: Co-expression modules in female mouse livers

12.2.1: Choosing the soft threshold beta via scale free topology

As detailed in section 4.3, we propose to choose the soft threshold power beta based on the scale free topology criterion Zhang and Horvath 2005, i.e. we choose the lowest beta that results in approximate scale free topology as measured by the scale free topology fitting index (Eq.~1.13) `R^2=ScaleFreeFit=cor(log(p(dk)),log(BinNo))^2`.

The following R code illustrates the use of the R function pickSoftThreshold for calculating scale free topology fitting indices `R^2` corresponding to different soft thresholding powers beta.

```
# Choose a set of soft thresholding powers
powers = c(1:10)  # in practice this should include powers up to 20.
# choose power based on SFT criterion
sft = pickSoftThreshold(datExprFemale, powerVector = powers)
```

```
##    Power SFT.R.sq  slope truncated.R.sq mean.k. median.k. max.k.
## 1      1   0.0286  0.350          0.461  747.00    762.00 1210.0
## 2      2   0.1240 -0.597          0.843  254.00    251.00  574.0
## 3      3   0.3390 -1.030          0.972  111.00    102.00  324.0
## 4      4   0.4990 -1.420          0.969   56.50     47.20  202.0
## 5      5   0.6770 -1.700          0.940   32.20     25.10  134.0
## 6      6   0.8990 -1.490          0.961   19.90     14.50   94.8
## 7      7   0.9200 -1.660          0.917   13.20      8.68   84.1
## 8      8   0.9040 -1.720          0.876    9.25      5.39   76.3
## 9      9   0.8620 -1.700          0.840    6.80      3.56   70.5
## 10    10   0.8360 -1.660          0.834    5.19      2.38   65.8
```

Digression: if you want to pick a soft threshold for a signed network write

```
sft=pickSoftThreshold(datExprFemale,powerVector=powers, networkType = "signed")
```

but then you should consider higher powers. Default `beta=12`.

```
# Plot the results:
par(mfrow = c(1, 2))
# SFT index as a function of different powers
plot(sft$fitIndices[, 1], -sign(sft$fitIndices[, 3]) * sft$fitIndices[, 2],
    xlab = "Soft Threshold (power)", ylab = "SFT, signed R^2", type = "n", main = paste("Sca
text(sft$fitIndices[, 1], -sign(sft$fitIndices[, 3]) * sft$fitIndices[, 2],
    labels = powers, col = "red")
# this line corresponds to using an R^2 cut-off of h
abline(h = 0.9, col = "red")
# Mean connectivity as a function of different powers
plot(sft$fitIndices[, 1], sft$fitIndices[, 5], type = "n", xlab = "Soft Threshold (power)",
    ylab = "Mean Connectivity", main = paste("Mean connectivity"))
text(sft$fitIndices[, 1], sft$fitIndices[, 5], labels = powers, col = "red")
```

Caption: Scale free topology criterion of the female mouse liver co-expression network. SFT plot for choosing the power beta for the unsigned weighted correlation network. Left hand side: the SFT index `R^2` (y-axis) as a function of different powers `beta` (x-axis). While R^2 tends to go up with higher powers, there is not a strictly monotonic relationship. Right hand side: the mean connectivity (`y`-axis) is a strictly decreasing function of the power `beta` (`x`-axis).

The result is shown in the Figure. We choose the power `beta=7` since this where the curve reaches a saturation point. Also note that for this choice of `beta`, we obtain the maximum value for `R^2=0.92`. Incidentally, this power is slightly different from the default choice (`beta=6`) for unsigned weighted networks. An advantage of weighted networks is that they are highly robust with regard to the power `beta`.

12.2.2 Automatic module detection via dynamic tree cutting

While the stepwise module detection approach described in section 12.2.4 allows the user to interact with the software it may be inconvenient. In contrast, the function blockwiseModules automatically implements all steps of module detection, i.e. it automatically constructs a correlation network, creates a cluster tree, defines modules as branches, and merges close modules. It outputs module colors and module eigengenes which can be used in subsequent analysis. Also one can visualize the results of the module detection. The function blockwiseModules has many parameters, and in this example most of them are left at their default value. We have attempted to provide reasonable default values, but they may not be appropriate for the particular data set the reader wishes to analyze. We encourage the user to read the help file provided within the package in the R environment and experiment with changing the parameter values.

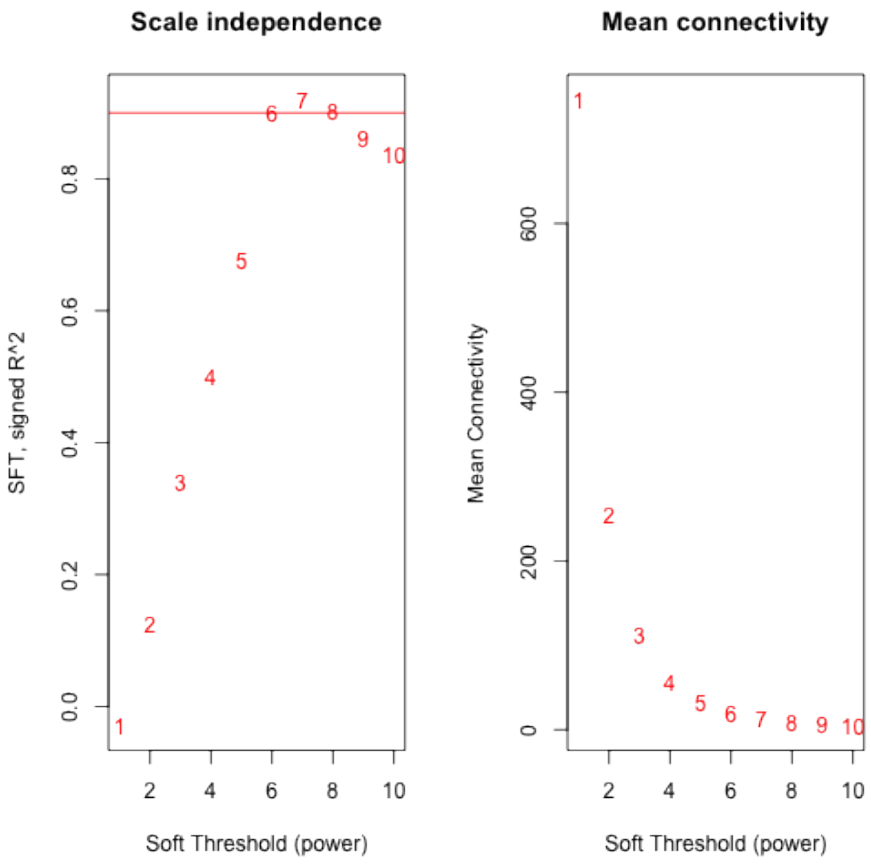

Figure 2: plot of chunk unnamed-chunk-11

```
mergingThresh = 0.25
net = blockwiseModules(datExprFemale, corType = "pearson", maxBlockSize = 5000,
    networkType = "unsigned", power = 7, minModuleSize = 30, mergeCutHeight = mergingThresh,
    numericLabels = TRUE, saveTOMs = TRUE, pamRespectsDendro = FALSE, saveTOMFileBase = "fem
moduleLabelsAutomatic = net$colors
# Convert labels to colors for plotting
moduleColorsAutomatic = labels2colors(moduleLabelsAutomatic)

# A data frame with module eigengenes can be obtained as follows
MEsAutomatic = net$MEs

# this is the body weight
weight = as.data.frame(datTraits$weight_g)
names(weight) = "weight"
# Next use this trait to define a gene significance variable
GS.weight = as.numeric(cor(datExprFemale, weight, use = "p"))
# This translates the numeric values into colors
GS.weightColor = numbers2colors(GS.weight, signed = T)
blocknumber = 1
datColors = data.frame(moduleColorsAutomatic, GS.weightColor)[net$blockGenes[[blocknumber]],
    ]

# Plot the dendrogram and the module colors underneath
plotDendroAndColors(net$dendrograms[[blocknumber]], colors = datColors, groupLabels = c("Mod
    "GS.weight"), dendroLabels = FALSE, hang = 0.03, addGuide = TRUE, guideHang = 0.05)
```

The result can be found in the Figure.

Caption. Hierarchical cluster tree (average linkage, dissTOM) of the 3600 genes. The color bands provide a simple visual comparison of module assignments (branch cuttings) based on the dynamic hybrid branch cutting method. The first band shows the results from the automatic single block analysis and the second color band visualizes the gene significance measure: `"red"` indicates a high positive correlation with mouse body weight. Note that the brown, blue and red module contain many genes that have high positive correlations with body weight.

The parameter maxBlockSize tells the function how large the largest block can be that the reader's computer can handle. The default value is 5000 which is appropriate for most modern desktops. Note that if this code were to be used to analyze a data set with more than 5000 rows, the function blockwiseModules would split the data set into several blocks. We briefly mention the function recutBlockwiseTrees that can be applied to the cluster tree(s) resulting from blockwiseModules. This function can be used to change the branch cutting parameters without having to repeat the computationally intensive recalculation of the cluster tree(s).

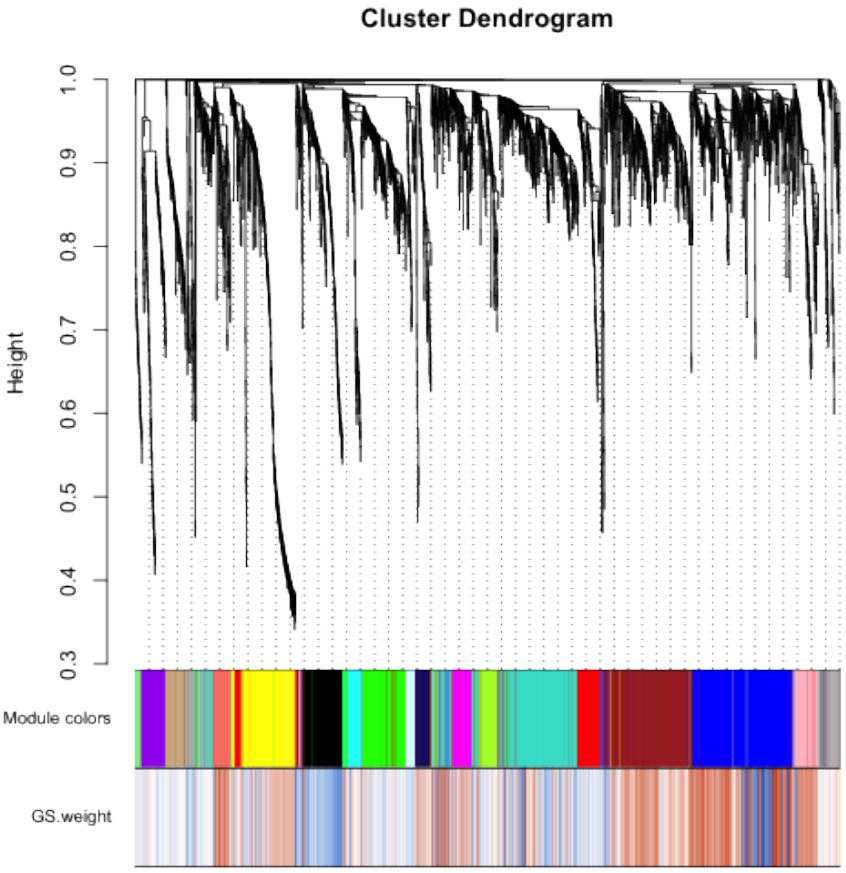

Figure 3: plot of chunk unnamed-chunk-12

12.2.3 Blockwise module detection for large networks

In this mouse co-expression network application, we work with a relatively small data set of 3600 measured probes. However, modern microarrays measure up to 50,000 probe expression levels at once. Constructing and analyzing networks with such large numbers of nodes is computationally challenging even on a large server. We now illustrate a method, implemented in the WGCNA package, that allows the user to perform a network analysis with such a large number of genes. Instead of actually using a very large data set, we will for simplicity pretend that hardware limitations restrict the number of genes that can be analyzed at once to 2000. The basic idea is to use a two-level clustering. First, we use a fast, computationally inexpensive and relatively crude clustering method to pre-cluster genes into blocks of size close to and not exceeding the maximum of 2000 genes. We then perform a full network analysis in each block separately. At the end, modules whose eigengenes are highly correlated are merged. The advantage of the block-wise approach is a much smaller memory footprint (which is the main problem with large data sets on standard desktop computers), and a significant speed-up of the calculations. The trade-off is that due to using a simpler clustering to obtain blocks, the blocks may not be optimal, causing some outlying genes to be assigned to a different module than they would be in a full network analysis.

We will now pretend that even the relatively small number of genes, 3600, that we have been using here is too large, and the computer we run the analysis on is not capable of handling more than 2000 genes in one block. The automatic network construction and module detection function blockwiseModules can handle the splitting into blocks automatically; the user just needs to specify the largest number of genes that can form a block:

```
bwnet = blockwiseModules(datExprFemale, corType = "pearson", maxBlockSize = 2000,
    networkType = "unsigned", power = 7, minModuleSize = 30, mergeCutHeight = mergingThresh,
    numericLabels = TRUE, saveTOMs = TRUE, pamRespectsDendro = FALSE, saveTOMFileBase = "fe
    verbose = 3)
```

```
##  Calculating module eigengenes block-wise from all genes
##    Flagging genes and samples with too many missing values...
##     ..step 1
##  ....pre-clustering genes to determine blocks..
##    Projective K-means:
##    ..k-means clustering..
##    ..merging smaller clusters...
## Block sizes:
## gBlocks
##    1    2
## 1870 1730
##  ..Working on block 1 .
```

```
##     Rough guide to maximum array size: about 46000 x 46000 array of doubles..
##     TOM calculation: adjacency..
##     ..will use 4 parallel threads.
##      Fraction of slow calculations: 0.429953
##     ..connectivity..
##     ..matrix multiply..
##     ..normalize..
##     ..done.
##    ..saving TOM for block 1 into file femaleMouseTOM-blockwise-block.1.RData
##  ....clustering..
##  ....detecting modules..
##  ....calculating module eigengenes..
##  ....checking modules for statistical meaningfulness..
##      ..removing 1 genes from module 1 because their KME is too low.
##  ..Working on block 2 .
##     Rough guide to maximum array size: about 46000 x 46000 array of doubles..
##     TOM calculation: adjacency..
##     ..will use 4 parallel threads.
##      Fraction of slow calculations: 0.359167
##     ..connectivity..
##     ..matrix multiply..
##     ..normalize..
##     ..done.
##    ..saving TOM for block 2 into file femaleMouseTOM-blockwise-block.2.RData
##  ....clustering..
##  ....detecting modules..
##  ....calculating module eigengenes..
##  ....checking modules for statistical meaningfulness..
##      ..removing 5 genes from module 1 because their KME is too low.
##      ..removing 1 genes from module 2 because their KME is too low.
##   ..reassigning 40 genes from module 1 to modules with higher KME.
##   ..reassigning 5 genes from module 2 to modules with higher KME.
##   ..reassigning 8 genes from module 3 to modules with higher KME.
##   ..reassigning 10 genes from module 4 to modules with higher KME.
##   ..reassigning 1 genes from module 5 to modules with higher KME.
##   ..reassigning 1 genes from module 7 to modules with higher KME.
##   ..reassigning 1 genes from module 8 to modules with higher KME.
##   ..reassigning 1 genes from module 11 to modules with higher KME.
##   ..reassigning 60 genes from module 15 to modules with higher KME.
##   ..reassigning 4 genes from module 16 to modules with higher KME.
##   ..reassigning 5 genes from module 17 to modules with higher KME.
##   ..reassigning 7 genes from module 18 to modules with higher KME.
##   ..reassigning 1 genes from module 19 to modules with higher KME.
##   ..reassigning 2 genes from module 21 to modules with higher KME.
##  ..merging modules that are too close..
##      mergeCloseModules: Merging modules whose distance is less than 0.25
```

```
##        Calculating new MEs...
```

Below we will compare the results of this analysis to the results of Section 12.2.2 in which all genes were analyzed in a single block. To make the comparison easier, we relabel the block-wise module labels so that modules with a significant overlap with single-block modules have the same label:

```
# Relabel blockwise modules so that their labels match those from our
# previous analysis
moduleLabelsBlockwise = matchLabels(bwnet$colors, moduleLabelsAutomatic)
# Convert labels to colors for plotting
moduleColorsBlockwise = labels2colors(moduleLabelsBlockwise)

# measure agreement with single block automatic procedure
mean(moduleLabelsBlockwise == moduleLabelsAutomatic)
```

```
## [1] 0.6989
```

The colorbands in the Figure below allow one to compare the results of the 2-block analysis with those from the single block analysis. Clearly, there is excellent agreement between the two automatic module detection methods.

The hierarchical clustering dendrograms (trees) used for the module identification in the `i`-th block are returned in `bwnet$dendrograms[[i]]`. For example, the cluster tree in the 2nd block can be visualized the following code:

```
blockNumber = 2
# Plot the dendrogram for the chosen block
plotDendroAndColors(bwnet$dendrograms[[blockNumber]], moduleColorsBlockwise[bwnet$blockGenes
    "Module colors", main = paste("Dendrogram and module colors in block", blockNumber),
    dendroLabels = FALSE, hang = 0.03, addGuide = TRUE, guideHang = 0.05)
```

The function recutBlockwiseTrees can be used to change parameters of the branch cutting procedure without having to recompute the cluster tree.

12.2.4 Manual, stepwise module detection

Here we present R code that implements the following steps of module detection using the default WGCNA approach. 1. Define a weighted adjacency matrix A. 2. Define the topological overlap matrix (Eq.1.27) based dissimilarity measure `dissTOM_{ij}=1-TopOverlap_{ij}`. 3. Construct a hierarchical cluster tree (average linkage). 4. Define modules as branches of the tree.

Branches of the dendrogram group together densely interconnected, highly co-expressed genes. Modules are defined as branches of the cluster tree, i.e. module

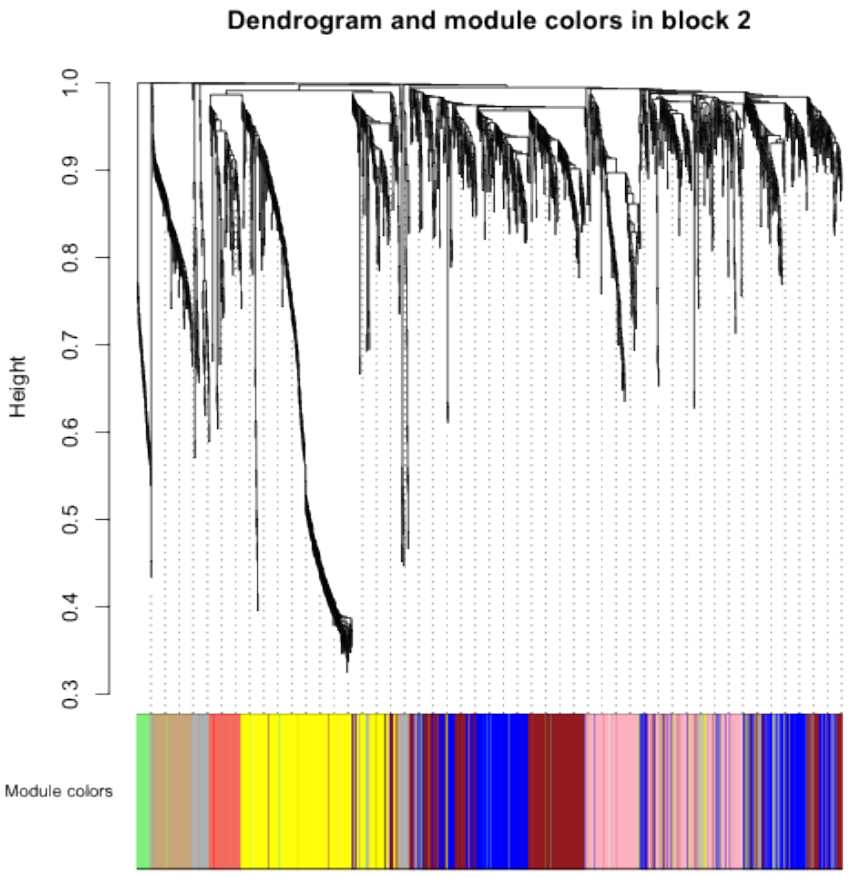

Figure 4: plot of chunk unnamed-chunk-15

detection involves cutting the branches of the tree. There are several methods for branch cutting. As described in section 8.6, the most flexible approach is the dynamic tree cutting method implemented in the cutreeDynamic R function (Langfelder, Zhang et al 2007).

Although the default values of the branch cutting method work well, the user may explore choosing alternative parameter values. In the following R code, we choose a relatively large minimum module size of `minClusterSize=30`, and a medium sensitivity (`deepSplit=2`) to branch splitting.

```
# We now calculate the weighted adjacency matrix, using the power 7:
A = adjacency(datExprFemale, power = 7)
# Digression: to define a signed network choose A =
# adjacency(datExprFemale, power = 12, type='signed')

# define a dissimilarity based on the topological overlap
dissTOM = TOMdist(A)
```

```
## ..connectivity..
## ..matrix multiply..
## ..normalize..
## ..done.
```

```
# hierarchical clustering
geneTree = flashClust(as.dist(dissTOM), method = "average")
# here we define the modules by cutting branches
moduleLabelsManual1 = cutreeDynamic(dendro = geneTree, distM = dissTOM, method = "hybrid",
    deepSplit = 2, pamRespectsDendro = F, minClusterSize = 30)
```

```
##  Detecting clusters...
##  ..cutHeight not given, setting it to 0.996  ===>  99% of the (truncated) height range in
```

```
# Relabel the manual modules so that their labels match those from our
# previous analysis
moduleLabelsManual2 = matchLabels(moduleLabelsManual1, moduleLabelsAutomatic)
# Convert labels to colors for plotting
moduleColorsManual2 = labels2colors(moduleLabelsManual2)
```

Clustering methods may identify modules whose expression profiles are very similar. More specifically, a module detection method may result in modules whose eigengenes are highly correlated. Since highly correlated modules are not distinct, it may be advisable to merge them. The following code shows how to create a cluster tree of module eigengenes and how to merge them (if their pairwise correlation is larger than `0.75`).

```
# Calculate eigengenes
MEList = moduleEigengenes(datExprFemale, colors = moduleColorsManual2)
MEs = MEList$eigengenes
# Add the weight to existing module eigengenes
MET = orderMEs(cbind(MEs, weight))
# Plot the relationships among the eigengenes and the trait
plotEigengeneNetworks(MET, "", marDendro = c(0, 4, 1, 2), marHeatmap = c(3,
    4, 1, 2), cex.lab = 0.8, xLabelsAngle = 90)
```

```
## Warning: WGCNA::greenWhiteRed: this palette is not suitable for people
## with green-red color blindness (the most common kind of color blindness).
## Consider using the function blueWhiteRed instead.
```

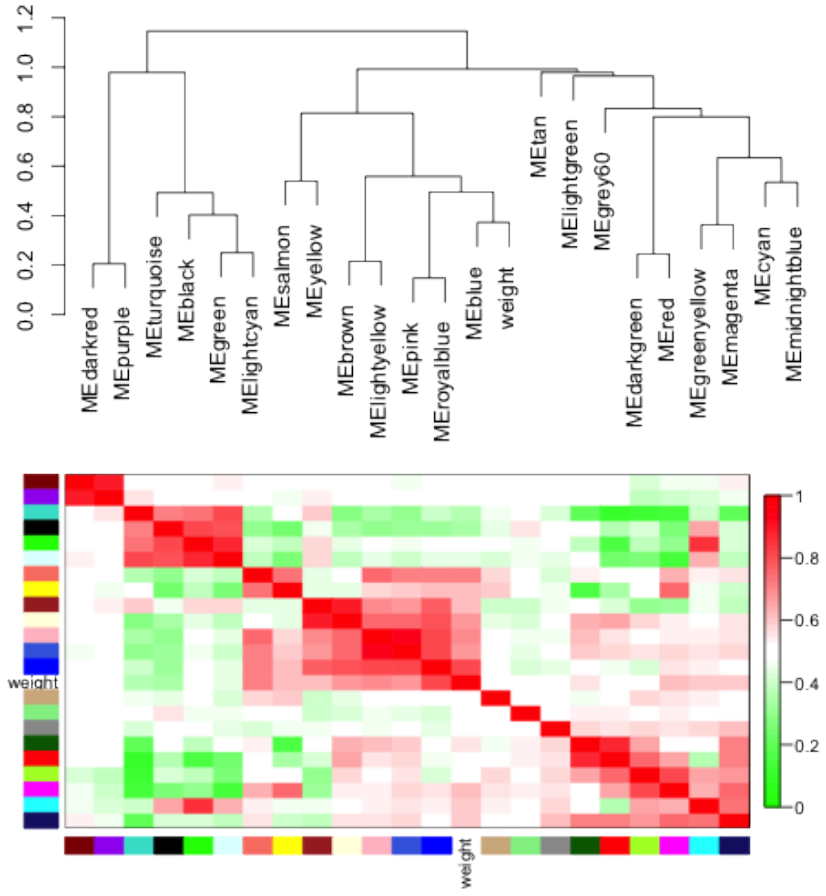

Figure 5: plot of chunk unnamed-chunk-17

The Figure shows the result of this analysis.

Caption: Visualization of the eigengene network representing the relationships among the modules and the sample trait body weight. The top panel shows a hierarchical clustering dendrogram of the eigengenes based on the dissimilarity diss(q_1,q_2)=1-cor(E^{(q_1)},E^{(q_2)}). The bottom panel shows the shows the eigengene adjacency A_{q1,q2}=0.5+0.5 cor(E^{(q_1)},E^{(q_2)}).

```
# automatically merge highly correlated modules
merge = mergeCloseModules(datExprFemale, moduleColorsManual2, cutHeight = mergingThresh)
```

```
##  mergeCloseModules: Merging modules whose distance is less than 0.25
##    Calculating new MEs...
```

```
# resulting merged module colors
moduleColorsManual3 = merge$colors
# eigengenes of the newly merged modules:
MEsManual = merge$newMEs
```

```
# Show the effect of module merging by plotting the original and merged
# module colors below the tree
datColors = data.frame(moduleColorsManual3, moduleColorsAutomatic, moduleColorsBlockwise,
    GS.weightColor)
plotDendroAndColors(geneTree, colors = datColors, groupLabels = c("manual hybrid",
    "single block", "2 block", "GS.weight"), dendroLabels = FALSE, hang = 0.03,
    addGuide = TRUE, guideHang = 0.05)
```

The result can be found in the Figure

Caption: Cluster tree of 3600 genes of the mouse liver co-expression network. The color bands provide a simple visual comparison of module assignments (branch cuttings) from 3 different branch cutting methods based on the dynamic hybrid branch cutting method. The first band shows the results from the manual (interactive) branch cutting approach, the second band shows the results of the automatic single block analysis, and the third band shows the results from the 2 block analysis. Note the high agreement among the methods. While overall little is lost when using the blockwise approach, there is some disagreement with respect to the blue module...

```
# check the agreement between manual and automatic module labels
mean(moduleColorsManual3 == moduleColorsAutomatic)
```

```
## [1] 0.8806
```

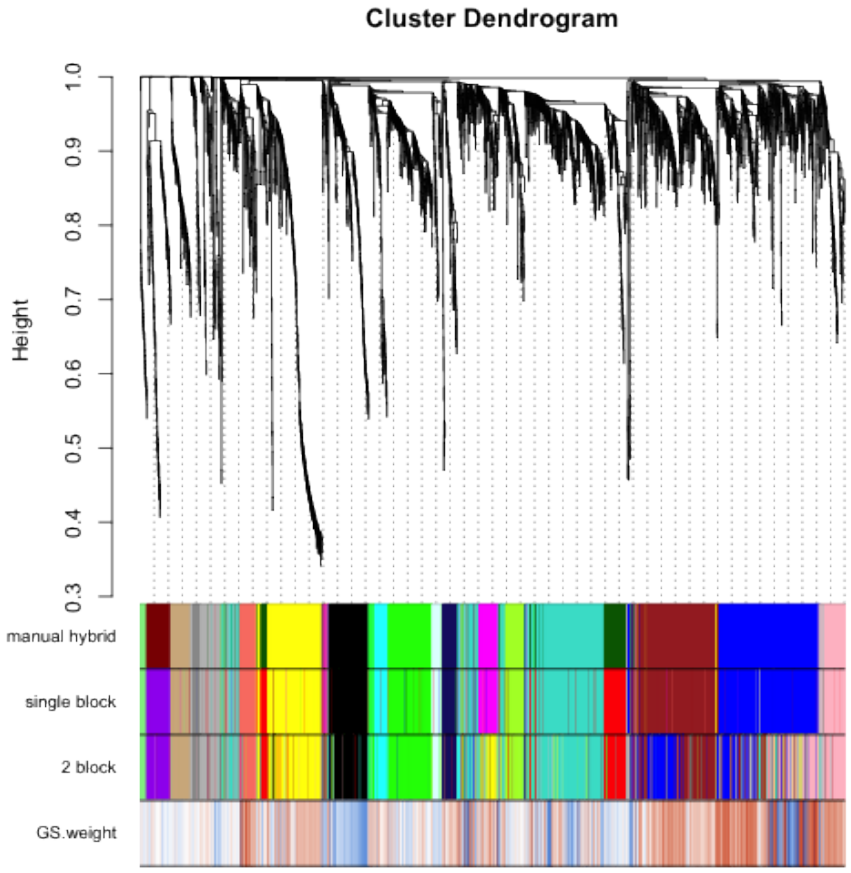

Figure 6: plot of chunk unnamed-chunk-18

12.2.5 Relating modules to physiological traits

Here we show how to identify modules that are significantly associated with the physiological traits. Since the module eigengene is an optimal summary of the gene expression profiles of a given module, it is natural to correlate eigengenes with these traits and to look for the most significant associations.

```
# Choose a module assignment
moduleColorsFemale = moduleColorsAutomatic
# Define numbers of genes and samples
nGenes = ncol(datExprFemale)
nSamples = nrow(datExprFemale)
# Recalculate MEs with color labels
MEs0 = moduleEigengenes(datExprFemale, moduleColorsFemale)$eigengenes
MEsFemale = orderMEs(MEs0)
modTraitCor = cor(MEsFemale, datTraits, use = "p")
modTraitP = corPvalueStudent(modTraitCor, nSamples)
# Since we have a moderately large number of modules and traits, a
# suitable graphical representation will help in reading the table. We
# color code each association by the correlation value: Will display
# correlations and their p-values
textMatrix = paste(signif(modTraitCor, 2), "\n(", signif(modTraitP, 1), ")",
    sep = "")
dim(textMatrix) = dim(modTraitCor)
par(mar = c(6, 8.5, 3, 3))
# Display the correlation values within a heatmap plot
labeledHeatmap(Matrix = modTraitCor, xLabels = names(datTraits), yLabels = names(MEsFemale),
    ySymbols = names(MEsFemale), colorLabels = FALSE, colors = greenWhiteRed(50),
    textMatrix = textMatrix, setStdMargins = FALSE, cex.text = 0.5, zlim = c(-1,
        1), main = paste("Module-trait relationships"))
```

```
## Warning: WGCNA::greenWhiteRed: this palette is not suitable for people
## with green-red color blindness (the most common kind of color blindness).
## Consider using the function blueWhiteRed instead.
```

The resulting color-coded table is shown in the Figure. The analysis identifies the several significant module-trait associations. We will concentrate on weight (first column) as the trait of interest.

Caption: Table of module-trait correlations and p-values. Each cell reports the correlation (and p-value) resulting from correlating module eigengenes (rows) to traits (columns). The table is color-coded by correlation according to the color legend.

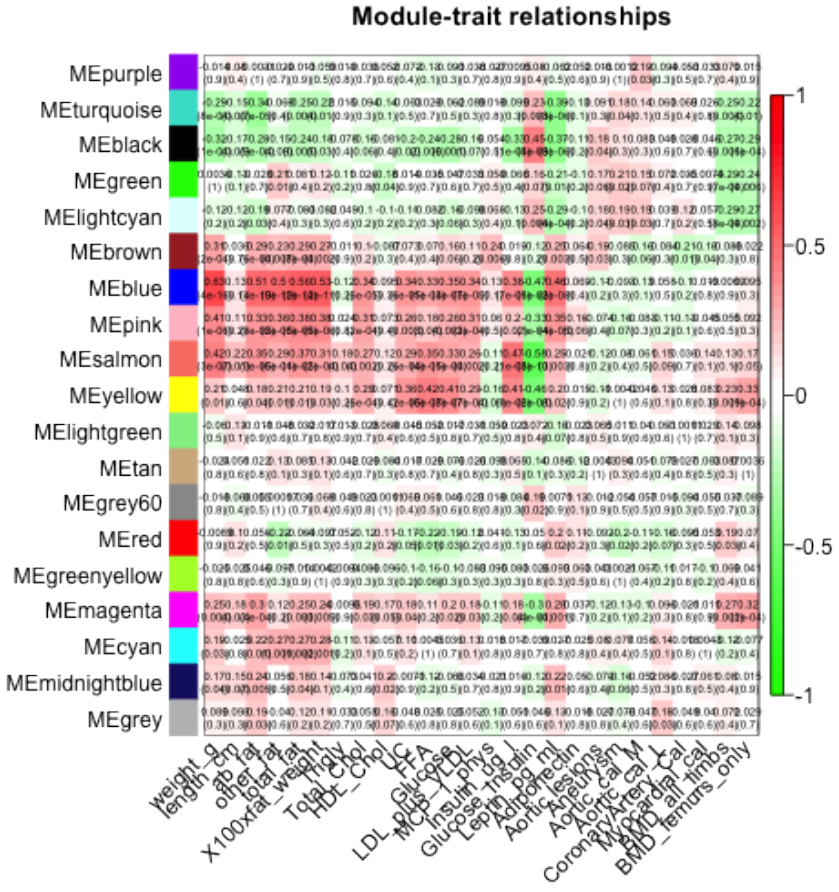

Figure 7: plot of chunk unnamed-chunk-20

Gene relationship to trait and important modules: Gene Significance and Module

Membership

We quantify associations of individual genes with our trait of interest (weight) by defining Gene Significance GS as (the absolute value of) the correlation between the gene and the trait. For each module, we also define a quantitative measure of module membership MM as the correlation of the module eigengene and the gene expression profile. This allows us to quantify the similarity of all genes on the array to every module.

```
# calculate the module membership values (aka. module eigengene based
# connectivity kME):
datKME = signedKME(datExprFemale, MEsFemale)
```

Intramodular analysis: identifying genes with high GS and MM

Using the GS and MM measures, we can identify genes that have a high significance for weight as well as high module membership in interesting modules. As an example, we look at the brown module that a high correlation with body weight. We plot a scatterplot of Gene Significance vs. Module Membership in select modules...

```
colorOfColumn = substring(names(datKME), 4)
par(mfrow = c(2, 2))
selectModules = c("blue", "brown", "black", "grey")
par(mfrow = c(2, length(selectModules)/2))
for (module in selectModules) {
    column = match(module, colorOfColumn)
    restModule = moduleColorsFemale == module
    verboseScatterplot(datKME[restModule, column], GS.weight[restModule], xlab = paste("Modu
        module, "module"), ylab = "GS.weight", main = paste("kME.", module,
        "vs. GS"), col = module)
}
```

Caption: Gene significance (`GS.weight`) versus module membership (`kME`) for the body weight related modules. `GS.weight` and `MM.weight` are highly correlated reflecting the high correlations between weight and the respective module eigengenes. We find that the brown, blue modules contain genes that have high positive and high negative correlations with body weight. In contrast, the grey "background" genes show only weak correlations with weight.

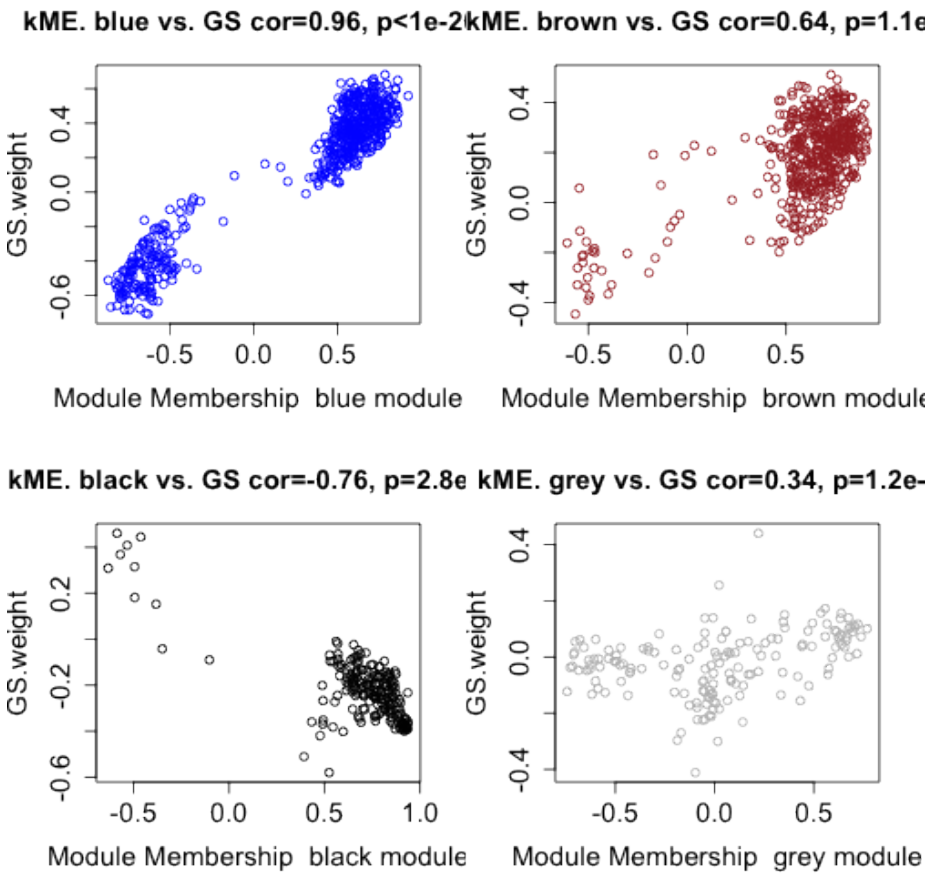

Figure 8: plot of chunk unnamed-chunk-22

12.2.6 Output file for gene ontology analysis

Here we present code for merging the probe data with a gene annotation table (which contains locus link IDs and other information for the probes). Next, we show how to export the results. The list of locus link identifiers (corresponding to the genes inside a given module) can be used as input of gene ontology (GO) and functional enrichment analysis software tools such as David (EASE) (http://david.abcc.ncifcrf.gov/) or AmiGO, Webgestalt. The R software also contains relevant packages, e.g. GOSim (Frohlich 2007). Most gene ontology based functional enrichment analysis software programs simply take lists of gene identifiers as input. Ingenuity Pathway Analysis allows the user to input gene expression data or gene identifiers.

The following code shows how to add gene identifiers and how to output the network analysis results.

```
# Read in the probe annotation
GeneAnnotation = read.csv(file.path(data.dir, "GeneAnnotation.csv"))
# Match probes in the data set to those of the annotation file
probes = names(datExprFemale)
probes2annot = match(probes, GeneAnnotation$substanceBXH)
# data frame with gene significances (cor with the traits)
datGS.Traits = data.frame(cor(datExprFemale, datTraits, use = "p"))
names(datGS.Traits) = paste("cor", names(datGS.Traits), sep = ".")
datOutput = data.frame(ProbeID = names(datExprFemale), GeneAnnotation[probes2annot,
    ], moduleColorsFemale, datKME, datGS.Traits)
# save the results in a comma delimited file
write.table(datOutput, "FemaleMouseResults.csv", row.names = F, sep = ",")
```

12.3 Systems genetic analysis with NEO

The identification of pathways and genes underlying complex traits using standard mapping techniques has been difficult due to genetic heterogeneity, epistatic interactions, and environmental factors. One promising approach to this problem involves the integration of genetics and gene expression.

Here, we describe a systems genetic approach for integrating gene co-expression network analysis with genetic data (Ghazalpour A, et al 2006,Presson A, et al 2008). The following steps summarize our overall approach: (1) Construct a gene co-expression network from gene expression data (see section 12.2.2) (2) Study the functional enrichment (gene ontology etc) of network modules (see section 12.2.6) (3) Relate modules to the clinical traits (see section 12.2.5) (4) Identify chromosomal locations (QTLs) regulating modules and traits (5) Use QTLs as causal anchors in network edge orienting analysis (described in chapter 11) to find causal drivers underlying the clinical traits.

In step 4, we identify chromosomal locations (genetic markers) that relate to body weight and module eigengenes. The traditional quantitative trait locus (QTL) mapping approach relates clinical traits (e.g. body weight) to genetic markers directly. To relate quantitative traits to a single nucleotide polymorphism (SNP) marker, one can use a correlation test (section 10.3)) if the SNP is numerically coded. For example, an additive marker coding of a SNP results in trivariate ordinal variable, which takes on values 0,1,2. Alternatively, one can carry out a LOD score analysis using the qtl R package (Broman et al 2003).

In practice, we find that results from a single locus LOD score analysis (with additive marker coding) are rank-equivalent with those from a correlation test analysis. Several authors have proposed a strategy that uses gene expression data to help map clinical traits. The strategy uses the genetics of gene expression and utilizes gene expression as a quantitative trait, thereby mapping expression QTL (eQTL) (Jansen et al 2001,Schadt et al 2003). Since we were interested in studying the genetics of modules as opposed to the genetics of the entire network, when searching for mQTLs, we focused on module-specific eQTL hot spots.

We used 1065 single nucleotide polymorphism (SNP) markers that were evenly spaced across the mouse genome, to map gene expression values for all genes within each gene module (Ghazalpour et al 2006). A module quantitative trait locus (mQTL) is a chromosomal location (e.g. a SNP) which correlates with many gene expression profiles of a given module.

We found that one of the modules that was highly correlated with body weight had nine module QTLs. These were located on Chromosomes 2, 3, 5, 10 (middle), 10 (distal), 12 (proximal), 12 (middle), 14, and 19. In particular, the mQTL on chromosome 19 has a highly significant correlation mouse body weight. We use this mQTL (denoted `mQTL19.047`) as causal anchor in step 5.

In step 5, we use genetic markers to calculate Local Edge Orienting (LEO) scores for a) determining whether the module eigengene has a causal effect on body weight, and b) for identifying genes inside the trait related modules that have a causal effect on body weight. For example, for the i-th expression profile x_i in the blue module, we calculate the LEO.SingleAnchor score (section 11.8.1) for assessing the relative causal fit of `mQTL19.047-> x_i ->weight` .

Copy and paste the following R code from section 11.8.1 into the R session.

```
suppressMessages(library(sem))
## Following does not work with knitr; best to put in separate file.  Now
## we specify the 5 single anchor models Single anchor model 1: M1->A->B
CausalModel1 = specifyModel(file.path(data.dir, "model1.txt"))
# Single anchor model 2: M->B->A
CausalModel2 = specifyModel(file.path(data.dir, "model2.txt"))
# Single anchor model 3: A<-M->B
CausalModel3 = specifyModel(file.path(data.dir, "model3.txt"))
# Single anchor model 4: M->A<-B
```

```
CausalModel4 = specifyModel(file.path(data.dir, "model4.txt"))
# Single anchor model 5: M->B<-A
CausalModel5 = specifyModel(file.path(data.dir, "model5.txt"))

# Function for obtaining the model fitting p-value from an sem object:
ModelFittingPvalue = function(semObject) {
    ModelChisq = summary(semObject)$chisq
    df = summary(semObject)$df
    ModelFittingPvalue = 1 - pchisq(ModelChisq, df)
    ModelFittingPvalue
}

# this function calculates the single anchor score
LEO.SingleAnchor = function(M1, A, B) {
    datObsVariables = data.frame(M1, A, B)
    # this is the observed correlation matrix
    S.SingleAnchor = cor(datObsVariables, use = "p")
    m = dim(datObsVariables)[[1]]

    semModel1 = sem(CausalModel1, S = S.SingleAnchor, N = m)
    semModel2 = sem(CausalModel2, S = S.SingleAnchor, N = m)
    semModel3 = sem(CausalModel3, S = S.SingleAnchor, N = m)
    semModel4 = sem(CausalModel4, S = S.SingleAnchor, N = m)
    semModel5 = sem(CausalModel5, S = S.SingleAnchor, N = m)

    # Model fitting p-values for each model
    p1 = ModelFittingPvalue(semModel1)
    p2 = ModelFittingPvalue(semModel2)
    p3 = ModelFittingPvalue(semModel3)
    p4 = ModelFittingPvalue(semModel4)
    p5 = ModelFittingPvalue(semModel5)
    LEO.SingleAnchor = log10(p1/max(p2, p3, p4, p5))

    data.frame(LEO.SingleAnchor, p1, p2, p3, p4, p5)
}  # end of function

# Get SNP markers which encode module QTLs
SNPdataFemale = read.csv(file.path(data.dir, "SNPandModuleQTLFemaleMice.csv"))
# find matching row numbers to line up the SNP data
snpRows = match(dimnames(datExprFemale)[[1]], SNPdataFemale$Mice)
# define a data frame whose rows correspond to those of datExpr
datSNP = SNPdataFemale[snpRows, -1]
rownames(datSNP) = SNPdataFemale$Mice[snpRows]
# show that row names agree
table(rownames(datSNP) == rownames(datExprFemale))
```

```
##
## TRUE
##  134
```

```
SNP = as.numeric(datSNP$mQTL19.047)
weight = as.numeric(datTraits$weight_g)
MEblue = MEsFemale$MEblue
MEbrown = MEsFemale$MEbrown

# evaluate the relative causal fit of SNP -> ME -> weight
LEO.SingleAnchor(M1 = SNP, A = MEblue, B = weight)[[1]]
```

```
## [1] -2.454
```

```
LEO.SingleAnchor(M1 = SNP, A = MEbrown, B = weight)[[1]]
```

```
## [1] -3.839
```

Thus, there is no evidence for a causal relationship ME-> weight if `mQTL19.047` is used as causal anchor. However, we caution the reader that other causal anchors (e.g. SNPs in other chromosomal locations) or other module eigengenes may lead to a different conclusion. The critical step in a NEO analysis is to find valid causal anchors. While the module eigengene has no causal effect on weight, it is possible that individual genes inside the brown module may have a causal effect. The following code shows how to identify genes inside the brown module that have a causal effect on weight.

```
whichmodule = "blue"
restGenes = moduleColorsFemale == whichmodule
datExprModule = datExprFemale[, restGenes]
attach(datExprModule)
LEO.SingleAnchorWeight = rep(NA, dim(datExprModule)[[2]])
for (i in c(1:dim(datExprModule)[[2]])) {
    ## printFlush(i)
    LEO.SingleAnchorWeight[i] = LEO.SingleAnchor(M1 = SNP, A = datExprModule[,
        i], B = weight)[[1]]
}

# Find gens with a very significant LEO score
restCausal = LEO.SingleAnchorWeight > 3  # could be lowered to 1
names(datExprModule)[restCausal]
```

```
## [1] "MMT00024300" "MMT00030448" "MMT00041314"
```

```
# this provides us the corresponding gene symbols
data.frame(datOutput[restGenes, c(1, 6)], LEO.SingleAnchorWeight = LEO.SingleAnchorWeight)[r
    ]
```

```
##           ProbeID   gene_symbol LEO.SingleAnchorWeight
## 7145  MMT00024300       Cyp2c40                 3.219
## 8873  MMT00030448 4833409A17Rik                 3.778
## 11707 MMT00041314          <NA>                 3.669
```

Note that the NEO analysis implicates 3 probes, one of which corresponds to the gene `Cyp2c40`. Using alternative mouse cross data, gene Cyp2c40 had already been been found to be related to body weight in rodents (reviewed in Ghazalpour et al 2006). Let's determine pairwise correlations

```
signif(cor(data.frame(SNP, MEblue, MMT00024300, weight), use = "p"), 2)
```

```
##                SNP MEblue MMT00024300 weight
## SNP           1.00   0.20       -0.70   0.32
## MEblue        0.20   1.00       -0.54   0.63
## MMT00024300 -0.70  -0.54        1.00  -0.53
## weight        0.32   0.63       -0.53   1.00
```

Note that the causal anchor (SNP) has a weak correlation (`r=0.2`) with MEblue, a strong negative correlation (`r=-0.70`) with probe `MMT00024300` (of gene `Cyp2c40`), and a positive correlation with weight (`r=0.32`). The differences in pairwise correlations illustrate why `MMT00024300` appears causal for weight while MEblue does not. Further, note that `MMT00024300` has only a moderately high module membership value of `kMEblue=-0.54`, i.e. this gene is not a hub gene in the blue module. This makes sense in light of our finding that the module eigengene (which can be interpreted as the ultimate hub gene) is not causal for body weight. In Presson et al 2008, we show how one can integrate module membership and causal testing analysis to develop systems genetic gene screening criteria.

12.4 Visualizing the network

Module structure and network connections can be visualized in several different ways. While R offers a host of network visualization functions, there are also valuable external software packages.

12.4.1 Connectivity-, TOM-, and MDS plots

A connectivity plot of a network is simply a heatmap representation of the connectivity patterns of the adjacency matrix or another measure of pairwise node interconnectedness. In an undirected network the connectivity plot is symmetrical around the diagonal and the rows and columns depict network nodes (which are arranged in the same order). Often the nodes are sorted to highlight the module structure. For example, when the topological overlap matrix is used as measure of pairwise interconnectedness, then the topological overlap matrix (TOM) plot (Ravasz et al 2002) is a special case of a connectivity plot. The TOM plot provides a 'reduced' view of the network that allows one to visualize and identify network modules. The TOM plot is a color-coded depiction of the values of the TOM measure whose rows and columns are sorted by the hierarchical clustering tree (which used the TOM-based dissimilarity as input).

As an example, consider the Figure below where red/yellow indicate high/low values of TOM_{ij}. Both rows and columns of TOM have been sorted using the hierarchical clustering tree. Since TOM is symmetric, the TOM plot is also symmetric around the diagonal. Since modules are sets of nodes with high topological overlap, modules correspond to red squares along the diagonal.

Each row and column of the heatmap corresponds to a single gene. The heatmap can depict topological overlap, with light colors denoting low adjacency (overlap) and darker colors higher adjacency (overlap).

In addition, the cluster tree and module colors are plotted along the top and left side of the heatmap. The Figure was created using the following code.

```
# WARNING: On some computers, this code can take a while to run (20 minutes??).
# I suggest you skip it.
# Set the diagonal of the TOM disscimilarity to NA
diag(dissTOM) = NA
# Transform dissTOM with a power to enhance visibility
TOMplot(dissim=dissTOM^7,dendro=geneTree,colors=moduleColorsFemale, main = "Network heatmap
```

Caption: Topological Overlap Matrix (TOM) plot (also known as connectivity plot) of the network connections. Genes in the rows and columns are sorted by the clustering tree. Light color represents low topological overlap and progressively darker red color represents higher overlap. Clusters correspond to squares along the diagonal. The cluster tree and module assignment are also shown along the left side and the top.

Multidimensional scaling (MDS) is an approach for visualizing pairwise relationships specified by a dissimilarity matrix. Each row of the dissimilarity matrix is visualized by a point in a Euclidean space.

The Euclidean distances between a pair of points is chosen to reflect the corresponding pairwise dissimilarity. An MDS algorithm starts with a dissimilarity

matrix as input and assigns a location to each row (object, node) in `d`-dimensional space, where d is specified a priori. If `d=2`, the resulting point locations can be displayed in the Euclidean plane.

There are two major types of MDS: classical MDS (implemented in the R function `cmdscale`) and non-metric MDS (R functions `isoMDS` and `sammon`). The following R code shows how to create a classical MDS plot using `d=2` scaling dimensions.

```
cmd1 = cmdscale(as.dist(dissTOM), 2)
par(mfrow = c(1, 1))
plot(cmd1, col = moduleColorsFemale, main = "MDS plot", xlab = "Scaling Dimension 1",
    ylab = "Scaling Dimension 2")
```

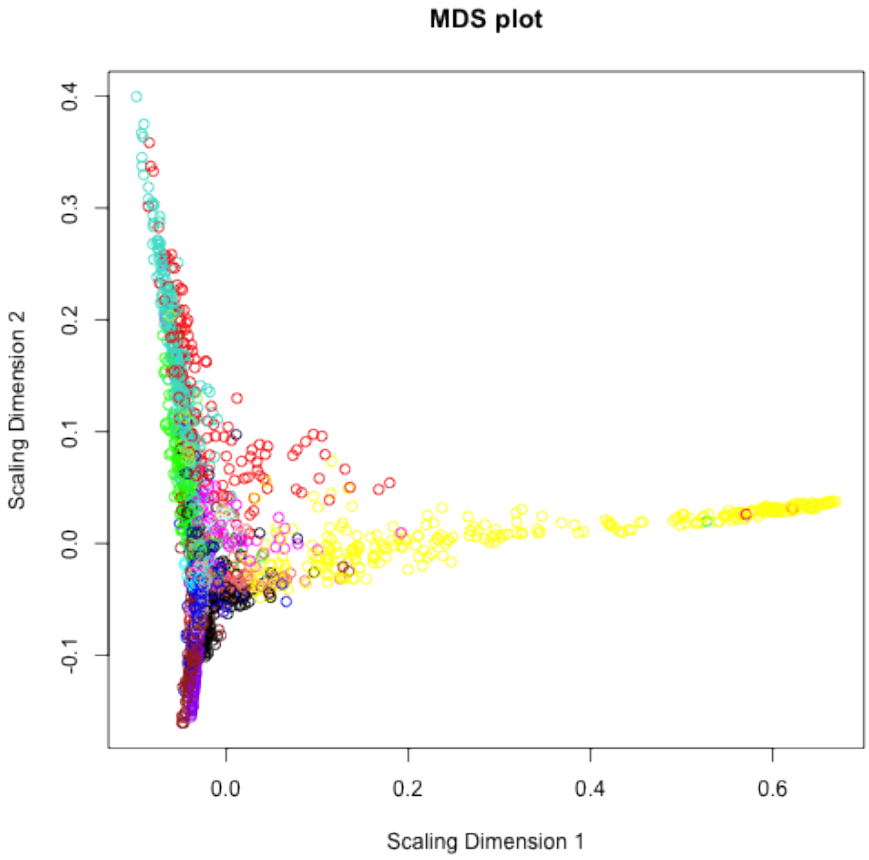

Figure 9: plot of chunk unnamed-chunk-29

The resulting plot can be found below.

Caption: Classical multidimensional scaling plot whose input is the TOM dissimilarity. Each dot (gene) is colored by the module assignment.

VisANT plot and software

VisANT (Hu et al 2004) is a software framework for visualizing, mining, analyzing and modeling multi-scale biological networks. The software can handle large-scale networks. It is also able to integrate interaction data, networks and pathways. The implementation of metagraphs enables VisANT to integrate multi-scale knowledge. The WGCNA R function exportNetworkToVisANT allows the user to export networks from R into a format suitable for ViSANT. The following R code shows how to create a VisANT input file for visualizing the intramodular connections of the blue module.

```
# Recalculate topological overlap
TOM = TOMsimilarityFromExpr(datExprFemale, power=7)
module = "blue"
# Select module probes
probes = names(datExprFemale)
inModule = (moduleColorsFemale==module)
modProbes = probes[inModule]
# Select the corresponding Topological Overlap
modTOM = TOM[inModule, inModule]
# Because the module is rather large,
# we focus on the 30 top intramodular hub genes
nTopHubs = 30
# intramodular connectivity
kIN = softConnectivity(datExprFemale[, modProbes])
selectHubs = (rank (-kIN) <= nTopHubs)
vis = exportNetworkToVisANT(modTOM[selectHubs,selectHubs],
  file=paste("VisANTInput-", module, "-top30.txt", sep=""),
  weighted=TRUE,threshold = 0, probeToGene=
  data.frame(GeneAnnotation$substanceBXH,GeneAnnotation$gene_symbol))
```

To provide an example of a VisANT visualization, you can load file produced by the above code in VisANT. For better readability we recommend you change the thresholds for displaying a link between two nodes.

Cytoscape and Pajek software

The WGCNA R function exportNetworkToCytoscape allows the user to export networks in a format suitable for Cytoscape (Shannon et al 2003). Cytoscape is

a widely used software platform for visualizing networks and biological pathways and integrating these networks with annotations, gene expression profiles and other data. Although Cytoscape was originally designed for biological research, it is a general platform for complex network analysis and visualization. Cytoscape core distribution provides a basic set of features for data integration and visualization. Additional features are (freely) available as plugins. For example, plugins are available for network and molecular profiling analyses, scripting, and connection with databases.

Cytoscape allows the user to input an edge file and a node file, which allow one to specify connection strenghts (link weights) and node colors. The following R code shows how to create an input file for Cytoscape.

```
# select modules
modules = c("blue","brown")
# Select module probes
inModule=is.finite(match(moduleColorsFemale,modules))
modProbes=probes[inModule]
match1=match(modProbes,GeneAnnotation$substanceBXH)
modGenes=GeneAnnotation$gene_symbol[match1]
# Select the corresponding Topological Overlap
modTOM = TOM[inModule, inModule]
dimnames(modTOM) = list(modProbes, modProbes)
# Export the network into edge and node list files for Cytoscape
cyt = exportNetworkToCytoscape(modTOM,
  edgeFile=paste("CytoEdge",paste(modules,collapse="-"),".txt",sep=""),
  nodeFile=paste("CytoNode",paste(modules,collapse="-"),".txt",sep=""),
  weighted = TRUE, threshold = 0.02,nodeNames=modProbes,
  altNodeNames = modGenes, nodeAttr = moduleColorsFemale[inModule])
```

Inputting the network into Cytoscape is a bit involved since many options exist for interpreting edges and nodes. We refer the reader to Cytoscape documentation for the necessary details. Another powerful network visualization software is called Pajek (de Nooy et al., 2005). This freely available software is widely used in many social network and biological applications.

12.5 Module preservation between female and male mice

In the following, we present R code for studying the preservation of gene co-expression modules between female and male mouse liver networks. In section 12.2, we present R code for defining the modules as branches of a hierarchical clustering tree. This application demonstrates that module preservation statistics allow us to identify invalid, non-reproducible modules due to array outliers. In this section we use the WGCNA function modulePreservation to study module preservation of female modules in the male data.

```
maleData = read.csv(file.path(data.dir, "LiverMaleFromLiverFemale3600.csv"))
names(maleData)[1:9]
```

```
## [1] "substanceBXH"   "gene_symbol"    "LocusLinkID"    "ProteomeID"
## [5] "cytogeneticLoc" "CHROMOSOME"     "StartPosition"  "EndPosition"
## [9] "F2_4"
```

```
datExprMale = data.frame(t(maleData[, -c(1:8)]))
names(datExprMale) = maleData$substanceBXH

# WARNING: On some computers, this code can take a while to run (30 minutes??).
# I suggest you skip it.

# We now set up the multi-set expression data
# and corresponding module colors:
setLabels = c("Female", "Male")
multiExpr=list(Female=list(data=datExprFemale),
           Male=list(data=datExprMale))
moduleColorsFemale=moduleColorsAutomatic
multiColor=list(Female=moduleColorsFemale)

# The number of permutations drives the computation time
# of the module preservation function. For a publication use 200 permutations.
# But for brevity, let's use a small number
nPermutations1=10
# Set it to a low number (e.g. 3) if only the medianRank statistic
# and other observed statistics are needed.
# Permutations are only needed for calculating Zsummary
# and other permutation test statistics.
# set the random seed of the permutation test analysis
set.seed(1)
system.time({
mp = modulePreservation(multiExpr, multiColor,
referenceNetworks = 1, nPermutations = nPermutations1,
randomSeed = 1, quickCor = 0, verbose = 3)
})
# Save the results of the module preservation analysis
save(mp, file = "modulePreservation.RData")
# If needed, reload the data:
load(file = "modulePreservation.RData")

# specify the reference and the test networks
ref=1; test = 2

Obs.PreservationStats= mp$preservation$observed[[ref]][[test]]
```

```
Z.PreservationStats=mp$preservation$Z[[ref]][[test]]
# Look at the observed preservation statistics
Obs.PreservationStats
# Z statistics from the permutation test analysis
Z.PreservationStats
```

Let us now visualize the data.

```
modColors = rownames(Obs.PreservationStats)
moduleSize = Obs.PreservationStats$moduleSize
# we will omit the grey module (background genes)
# and the gold module (random sample of genes)
selectModules = !(modColors %in% c("grey", "gold"))
# Text labels for points
point.label = modColors[selectModules]

#Composite preservation statistics
medianRank=Obs.PreservationStats$medianRank.pres
Zsummary=Z.PreservationStats$Zsummary.pres

par(mfrow=c(1,2),mar = c(4.5,4.5,2.5,1))
# plot medianRank versus module size
plot(moduleSize[selectModules],medianRank[selectModules],col=1,
bg=modColors[selectModules],pch = 21,main="medianRank Preservation",
cex = 2, ylab ="medianRank",xlab="Module size", log="x")
labelPoints(moduleSize[selectModules],medianRank[selectModules],point.label,cex=1,offs=0.03)

# plot Zsummary versus module size
plot(moduleSize[selectModules],Zsummary[selectModules], col = 1,
bg=modColors[selectModules],pch = 21,main="Zsummary Preservation",
cex=2,ylab ="Zsummary", xlab = "Module size", log = "x")
labelPoints(moduleSize[selectModules],Zsummary[selectModules],point.label,cex=1,offs=0.03)
# Add threshold lines for Zsummary
abline(h=0); abline(h=2, col = "blue", lty = 2); abline(h=10, col = "red", lty = 2)
```

Caption: Preservation of female mouse liver modules in male livers. The right panel shows the composite statistic medianRank (Eq.9.20) versus module size. The higher the medianRank the less preserved is the module relative to other modules. Since medianRank is based on the observed preservation statistics (as opposed to Z statistics or p-values) we find that it is much less dependent on module size. The upper right panel shows the composite statistic Zsummary (Eq.9.1). If Zsummary$>$ 10 there is strong evidence that the module is preserved (Langfelder et al 2011). If Zsummary$<$2, there is no evidence that the module preserved. Note that Zsummary shows a strong dependence on module size. The lightgreen female module shows no evidence of preservation in the male liver network.

The resulting plot is shown in the Figure. We find that most modules show strong evidence of preservation Zsummary>10. The exception is the lightgreen module which has the second highest medianRank statistic implying that all but one of the other modules show stronger evidence of preservation. The lightgreen module also shows no evidence of preservation according to the Zsummary statistic for which it obtains the lowest value of 2.00.

Let us now create a heatmap of the lightgreen module expression values.

```
whichmodule = "lightgreen"
Eigengene = MEsFemale$MElightgreen
datExprModule = datExprFemale[, moduleColorsFemale == whichmodule]
# set the margins of the graphics window
par(mfrow = c(1, 1), mar = c(0.3, 5.5, 3, 2))
# create a heatmap whose columns correspond to the arrays and whose rows
# correspond to genes
plotMat(t(scale(datExprModule)), cex.axis = 2, nrgcols = 30, rlabels = F, rcols = whichmodul
    main = paste("heatmap", whichmodule, "module"))
```

Caption: The panel shows a heatmap of the `lighgreen` module gene expressions (rows correspond to genes, columns correspond to female mouse tissue samples). The heatmap color-codes the scaled gene expression values: red corresponds to over-expression and green to under-expression. Note that most genes are under-expressed in a single female mouse, which suggests that this module is due to an array outliers. Let us now produce a scatter plot of module eigengene of the light green module versus the standardized sample network connectivity `Z.k` (Eq. 7.24), which is the measure of outlyingness described in section 12.1.

```
# scatter plot between eigengene and sample network connectivity
par(mfrow = c(1, 1))
verboseScatterplot(Eigengene, Z.k, xlab = paste("ME", whichmodule, sep = ""))
abline(h = -2, col = "red", lwd = 2)
```

Caption: Scatter plot of the lightgreen module eigengene (`x`-axis) versus the standardized sample network connectivity `Z.k` (Eq.7.24), which is the measure of outlyingness described in section 12.1. Note that the module eigengene takes on an extreme negative value for one female mouse sample (corresponding to the sample where most module genes are under-expressed). The red horizontal line in the scatter plot corresponds to a `Z.k` threshold of `-2`.

Note that the eigenvector takes on an extreme negative value for one female mouse sample (corresponding to the sample where most module genes are under-expressed). Note that the outlying female mouse sample would have been removed if we had applied a stringent threshold of `-2` to `Z.k`. This supports the golden rule of data pre-processing: when in doubt, throw it out.

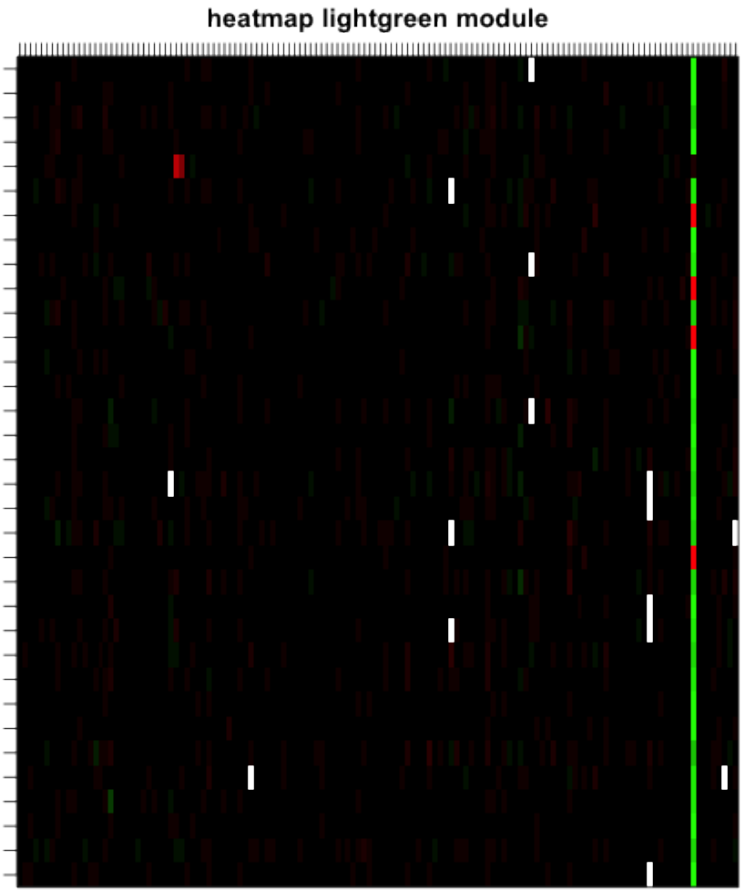

Figure 10: plot of chunk unnamed-chunk-31

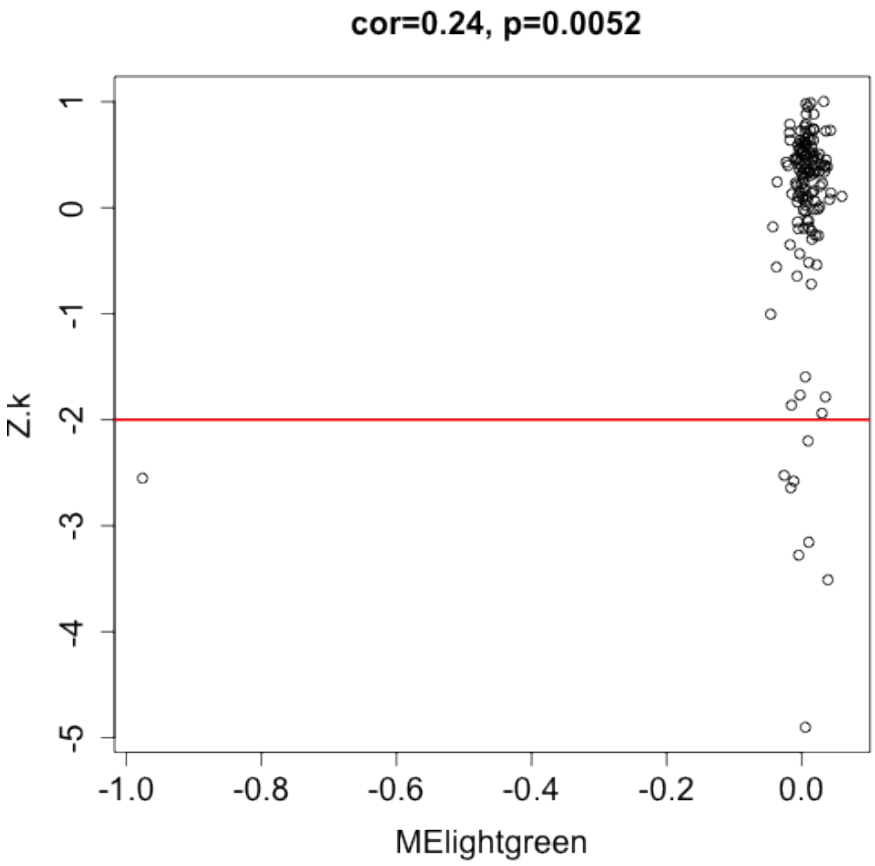

Figure 11: plot of chunk unnamed-chunk-32

In section 8.7, we described several cluster quality statistics based on network concepts. In Table 8.1, we provide an verview of the cluster quality statistics (Eq.8.12). In the following we present R code for evaluating the cluster quality of the modules inside the female mouse network.

```
# This calculates the quality of the modules in the female data
Obs.QualityStatsFemale= mp$quality$observed[[1]][[2]]
Z.QualityStatsFemale=mp$quality$Z[[1]][[2]]
QualityStats=data.frame(Obs.QualityStatsFemale[,1:2],
Zsummary.qual=Z.QualityStatsFemale$Zsummary.qual)
QualityStats["lightgreen",]
```

According to the quality statistics in the female network, the lightgreen module looks fine (medianRank.qual=1 ,Zsummary.qual= 34.7). This illustrates that no matter how good the quality statistics of a module in reference data set, there is no guarantee that it will be preserved in a test data set.

12.6 Consensus modules between female and male liver tissues

In section 12.5, we described how to assess module preservation between female and male mice. A related but distinct data analysis task is to construct modules that are present in several networks. In section 7.11.1 we described how to construct "consensus" modules that are present in each of the underlying networks. For example, consensus modules in the female and male mouse liver networks are trivially preserved in each sex. In the following, we describe how to use the consensus dissimilarity D^{Consensus,TOM} (Eq 7.40) for finding modules that are present in multiple correlation networks. Specifically, we find consensus modules that are present in both the female and the male mouse liver networks. We start out by defining the input of the automatic module detection function blockwiseConsensusModules.

```
# number of networks used in the consensus network analysis:
nSets = 2
# Vector with descriptive names of the two sets.
setLabels = c("Female liver", "Male liver")
shortLabels = c("Female", "Male")
# Define a list whose components contain the data
multiExpr = vector(mode = "list", length = nSets)
multiExpr[[1]] = list(data = datExprFemale)
names(multiExpr[[1]]$data) = names(datExprFemale)
rownames(multiExpr[[1]]$data) = dimnames(datExprFemale)[[1]]
multiExpr[[2]] = list(data = datExprMale)
names(multiExpr[[2]]$data) = names(datExprMale)
```

```
rownames(multiExpr[[2]]$data) = dimnames(datExprMale)[[1]]
# Check that the data has the correct format:
exprSize = checkSets(multiExpr)

# The variable exprSize contains useful information about the sizes of all
# of the data sets now we run automatic module detection procedure
netConsensus = blockwiseConsensusModules(multiExpr, maxBlockSize = 5000, power = 7,
    minModuleSize = 30, deepSplit = 2, pamRespectsDendro = FALSE, mergeCutHeight = 0.25,
    numericLabels = TRUE, minKMEtoStay = 0, saveTOMs = TRUE)
```

```
##  Calculating consensus modules and module eigengenes block-wise from all genes
##  Calculating consensus topological overlaps block-wise from all genes
##    Flagging genes and samples with too many missing values...
##     ..step 1
##     Rough guide to maximum array size: about 46000 x 46000 array of doubles..
##     TOM calculation: adjacency..
##     ..will use 4 parallel threads.
##      Fraction of slow calculations: 0.396405
##     ..connectivity..
##     ..matrix multiply..
##     ..normalize..
##     ..done.
##     Rough guide to maximum array size: about 46000 x 46000 array of doubles..
##     TOM calculation: adjacency..
##     ..will use 4 parallel threads.
##      Fraction of slow calculations: 0.438385
##     ..connectivity..
##     ..matrix multiply..
##     ..normalize..
##     ..done.
##  ..Working on block 1 .
##  ..merging consensus modules that are too close..
```

```
# list of consensus module eigengenes
consMEs = netConsensus$multiMEs
# module labels
modLabCons0 = netConsensus$colors
# Relabel the consensus modules so that their labels match those from the
# automatic analysis in female mice only
modLabCons = matchLabels(modLabCons0, moduleLabelsAutomatic)
# check the agreement between consensus- and females-only modules
mean(modLabCons == moduleLabelsAutomatic)
```

```
## [1] 0.6236
```

```
# Convert the numeric labels to color labels
moduleColorsConsensus = labels2colors(modLabCons)
```

Similar to the function blockwiseModules, blockwiseConsensusModules has many parameters, and in this example most of them are left at their default value. When dealing with many genes, it can be advantageous to carry out a blockwise consensus module detection analysis analogous to the case of a single network (see section 12.2.3).

The function `blockwiseConsensusModules` automatically partitions the genes into blocks of maximum size specified by the argument `maxBlockSize`. Since in our case, the number of genes is smaller than `maxBlockSize` the analysis amounts to a single block analysis. The consensus dendrogram (of the first block) is given by

```
blocknumber = 1
consTree = netConsensus$dendrograms[[blocknumber]]
# plot the consensus dendrogram and module colors
datColors = data.frame(moduleColorsConsensus, moduleColorsFemale)[netConsensus$blockGenes[[b
    ]
plotDendroAndColors(consTree, datColors, c("Cons module", "female module"),
    dendroLabels = FALSE, hang = 0.03, addGuide = TRUE, guideHang = 0.05, main = "Consensus
```

The Figure shows the consensus dendrogram, modules, and a comparison with the female modules.

Caption: The consensus dendrogram for female and male mouse liver coexpression networks. The average linkage hierarchical clustering tree is based on the consensus dissimilarity `D^{Consensus,TOM}` (Eq.7.40) of the female and male data. The first color-band shows the result of dynamic hybrid branch cutting of the consensus dendrogram. The second color-band shows the module colors based on the female mouse data alone. By visual inspection, it is clear that there is high agreement between the two module assignments, which reflects that many of the female mouse modules are also preserved in the male network.

12.6.1 Relating consensus modules to the traits

In this section we illustrate the use of module eigengenes to relate consensus modules to external microarray sample information such as classical clinical traits. In this analysis we have available several clinical traits. We relate the traits to consensus module eigengenes in each of the two sets. We remind the reader that while the consensus modules is a single module assignment for all genes, the module eigengenes represent the modules in each of the two sets. In other words, we have a single module assignment for each gene, but we have two sets of consensus module eigengenes, because a given module (set of genes) has a

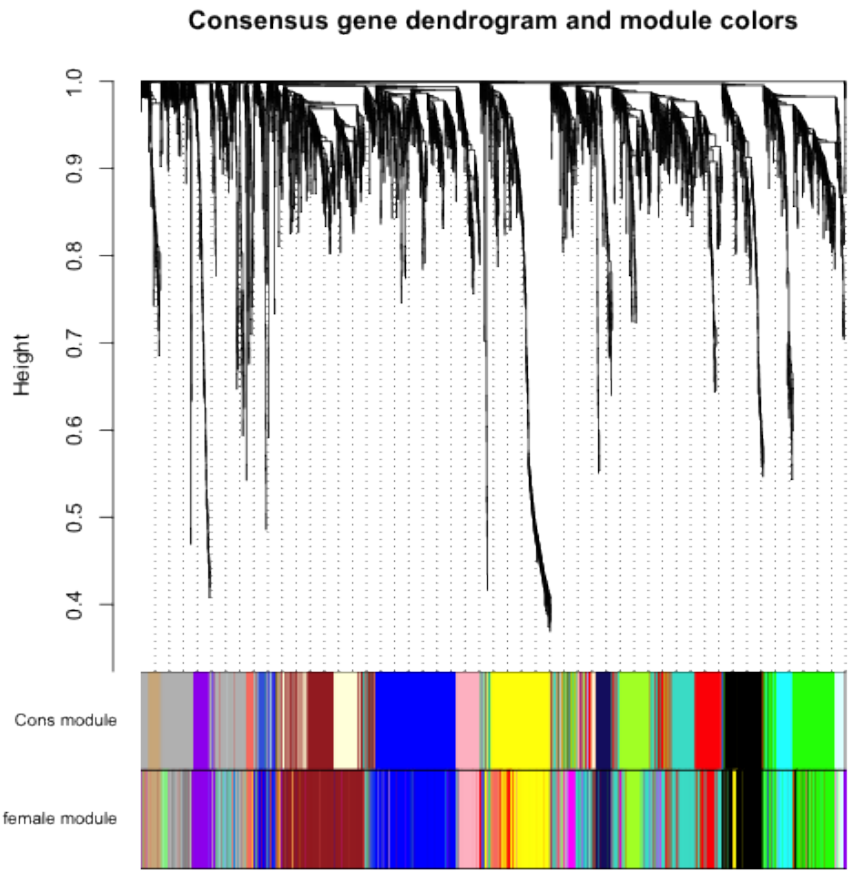

Figure 12: plot of chunk unnamed-chunk-34

particular expression profile in the female mice, and a different expression profile in the male mice. Similarly, we have the trait data separately for the female and for the male mice.

```
traitData = read.csv("ClinicalTraits.csv")
```

```
## Warning: cannot open file 'ClinicalTraits.csv': No such file or directory
```

```
## Error: cannot open the connection
```

```
dim(traitData)
```

```
## [1] 361  38
```

```
names(traitData)
```

```
##  [1] "X"                 "Mice"              "Number"
##  [4] "Mouse_ID"          "Strain"            "sex"
##  [7] "DOB"               "parents"           "Western_Diet"
## [10] "Sac_Date"          "weight_g"          "length_cm"
## [13] "ab_fat"            "other_fat"         "total_fat"
## [16] "comments"          "X100xfat_weight"   "Trigly"
## [19] "Total_Chol"        "HDL_Chol"          "UC"
## [22] "FFA"               "Glucose"           "LDL_plus_VLDL"
## [25] "MCP_1_phys"        "Insulin_ug_l"      "Glucose_Insulin"
## [28] "Leptin_pg_ml"      "Adiponectin"       "Aortic.lesions"
## [31] "Note"              "Aneurysm"          "Aortic_cal_M"
## [34] "Aortic_cal_L"      "CoronaryArtery_Cal" "Myocardial_cal"
## [37] "BMD_all_limbs"     "BMD_femurs_only"
```

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

```
## [1] 361  27
```

```
names(allTraits)
```

```
##  [1] "Mice"              "weight_g"          "length_cm"
##  [4] "ab_fat"            "other_fat"         "total_fat"
##  [7] "X100xfat_weight"   "Trigly"            "Total_Chol"
```

```
## [10] "HDL_Chol"           "UC"                "FFA"
## [13] "Glucose"            "LDL_plus_VLDL"     "MCP_1_phys"
## [16] "Insulin_ug_l"       "Glucose_Insulin"   "Leptin_pg_ml"
## [19] "Adiponectin"        "Aortic.lesions"    "Aneurysm"
## [22] "Aortic_cal_M"       "Aortic_cal_L"      "CoronaryArtery_Cal"
## [25] "Myocardial_cal"     "BMD_all_limbs"     "BMD_femurs_only"
```

```
# Form a multi-set structure that will hold the clinical traits.
TraitsL = vector(mode = "list", length = nSets)
for (set in 1:nSets) {
    setSamples = rownames(multiExpr[[set]]$data)
    traitRows = match(setSamples, allTraits$Mice)
    TraitsL[[set]] = list(data = allTraits[traitRows, -1])
    rownames(TraitsL[[set]]$data) = allTraits[traitRows, 1]
}
collectGarbage()

# Define data set dimensions
nGenes = exprSize$nGenes
nSamples = exprSize$nSamples

# Set up variables to contain the module-trait correlations
modTraitCor = list()
modTraitP = list()
# Calculate the correlations
for (set in 1:nSets) {
    modTraitCor[[set]] = cor(consMEs[[set]]$data, TraitsL[[set]]$data, use = "p")
    modTraitP[[set]] = corPvalueFisher(modTraitCor[[set]], exprSize$nSamples[set])
}
```

Now we will display the module-trait relationships using a color-coded table. The table reports the correlations and the corresponding p-values. Entries are color-coded according to the p-value.

```
MEColors = substring(names(consMEs[[1]]$data), 3)
MEColorNames = paste("ME", MEColors, sep = "")
# Plot the module-trait relationship table for set number
set = 1  # 1 for females, 2 for males
textMatrix = paste(signif(modTraitCor[[set]], 2), "\n(", signif(modTraitP[[set]],
    1), ")", sep = "")
dim(textMatrix) = dim(modTraitCor[[set]])
par(mar = c(6, 8.8, 3, 2.2))
labeledHeatmap(Matrix = modTraitCor[[set]], xLabels = names(TraitsL[[set]]$data),
```

```
    yLabels = MEColorNames, ySymbols = MEColorNames, colorLabels = FALSE, colors = greenWhit
    textMatrix = textMatrix, setStdMargins = FALSE, cex.text = 0.5, zlim = c(-1,
        1), main = paste("Module-trait relationships in", setLabels[set]))
```

```
## Warning: WGCNA::greenWhiteRed: this palette is not suitable for people
## with green-red color blindness (the most common kind of color blindness).
## Consider using the function blueWhiteRed instead.
```

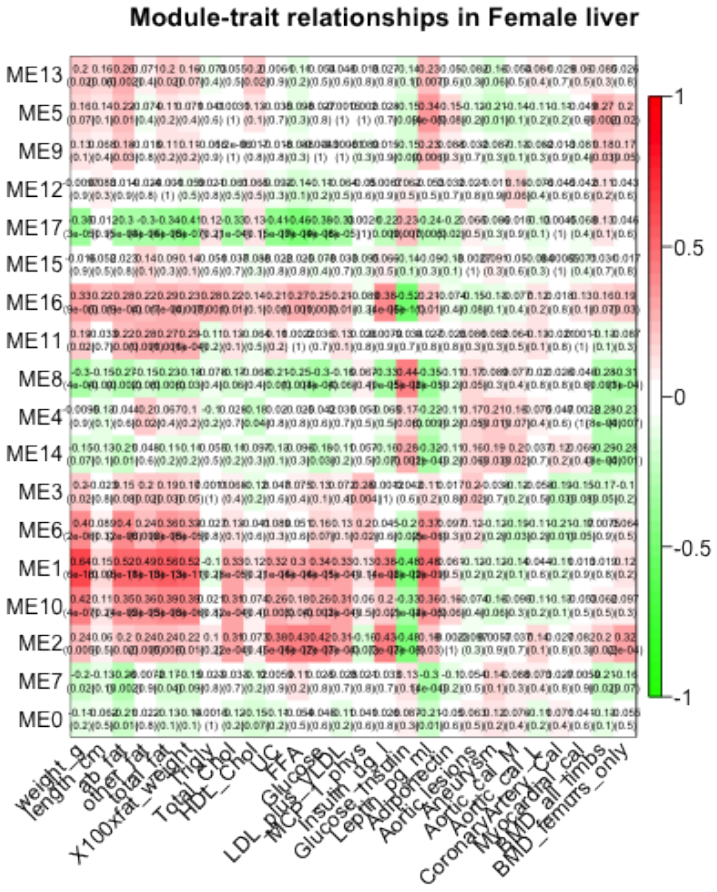

Figure 13: plot of chunk unnamed-chunk-38

Caption: Table of correlations and p-values for studying the relationships between consensus module eigengenes (rows) and physiological traits (columns) in the female mice. An analogous table could also be created for the male mice. The Figure presents the color-coded table. One can create an analogous Figure based on the male data by choosing set=2 in the above code.

12.6.2 Manual consensus module analysis

Here we present a manual or interactive approach for finding consensus modules.

Calculation of network adjacencies Network construction starts by calculating the adjacencies in the individual sets

```
beta = 7
# Initialize an appropriate array to hold the adjacencies
adjacencies = array(0, dim = c(nSets, nGenes, nGenes))
# Calculate adjacencies in each individual data set
for (set in 1:nSets) {
    adjacencies[set, , ] = abs(cor(multiExpr[[set]]$data, use = "p"))^beta
}

# Let's calculate corresponding Topological Overlap Matrices: Initialize
# an appropriate array to hold the TOMs
TOM = array(0, dim = c(nSets, nGenes, nGenes))
# Calculate TOMs in each individual data set
for (set in 1:nSets) {
    TOM[set, , ] = TOMsimilarity(adjacencies[set, , ])
}
```

```
## ..connectivity..
## ..matrix multiply..
## ..normalize..
## ..done.
## ..connectivity..
## ..matrix multiply..
## ..normalize..
## ..done.
```

```
# remove the adjacencies to free up space
rm(adjacencies)
```

Since the topological overlap matrices of the individual data sets may be differently calibratedd (e.g. reflecting different sample sizes) it can be advantageous to calibrated them. For example, the TOM in the male data may be systematically lower than the TOM in female data. Since consensus is defined as the componentwise minimum of the two TOMs, a bias may result. Here we illustrate a simple scaling that mitigates the effect of different statistical properties to some degree. In section 4.5, we describe how to use the power adjacency function `AF^{power}` (Eq. 4.2) to calibrate weighted networks. We calibrate the male TOM such that its `rob=0.95` quantile equals that of the female TOM. The notation is explained in section 4.5.

```
# Define the percentile (called prob) for defining the quantile
prob = 0.95
# Set random seed for reproducibility of sampling
set.seed(1)
# number of randomly samples entries of the TOM matrix
nSubset = 20000
# Choose the sampled TOM entries
subsetEntries = sample(nGenes * (nGenes - 1)/2, size = nSubset)
TOMsubset = list()
# vector of quantiles of the individual TOM matrices
quantile.TOM = rep(1, nSets)
# Scaling powers to equalize reference TOM values
beta.prob = rep(1, nSets)
# Loop over sets
for (set in 1:nSets) {
    # Select the sampled TOM entries
    TOMsubset[[set]] = vectorizeMatrix(TOM[set, , ])[subsetEntries]
    # Calculate the quantile corresponding to prob
    quantile.TOM[set] = quantile(TOMsubset[[set]], probs = prob, type = 8)
    # calculate the power of the adjacency function
    beta.prob[set] = log(quantile.TOM[1])/log(quantile.TOM[set])
    # use the power adjacency function for calibrating the TOM
    TOM[set, , ] = TOM[set, , ]^(beta.prob[set])
}
# let us look at the powers used for calibration
beta.prob
```

```
## [1] 1.0000 0.9799
```

```
# Pairwise relationship between uncalibrated matrices
par(mfrow = c(1, 1))
verboseScatterplot(TOMsubset[[1]], TOMsubset[[2]], xlab = "TOM[[1]]", ylab = "TOM[[2]]")
abline(0, 1)
```

By definition, the power beta of the reference (female) network equals 1. Note that the power beta.95=0.98 for the test (male) network is very close to 1 suggesting that even without calibration, the two networks are very comparable.

Consensus module identification We now calculate the consensus Topological Overlap by defining it using the parallel minimum of the TOMs in individual sets (Eq. 7.36) We use the consensus TOM as input to hierarchical clustering, and identify modules in the resulting dendrogram using the Dynamic Tree Cut algorithm.

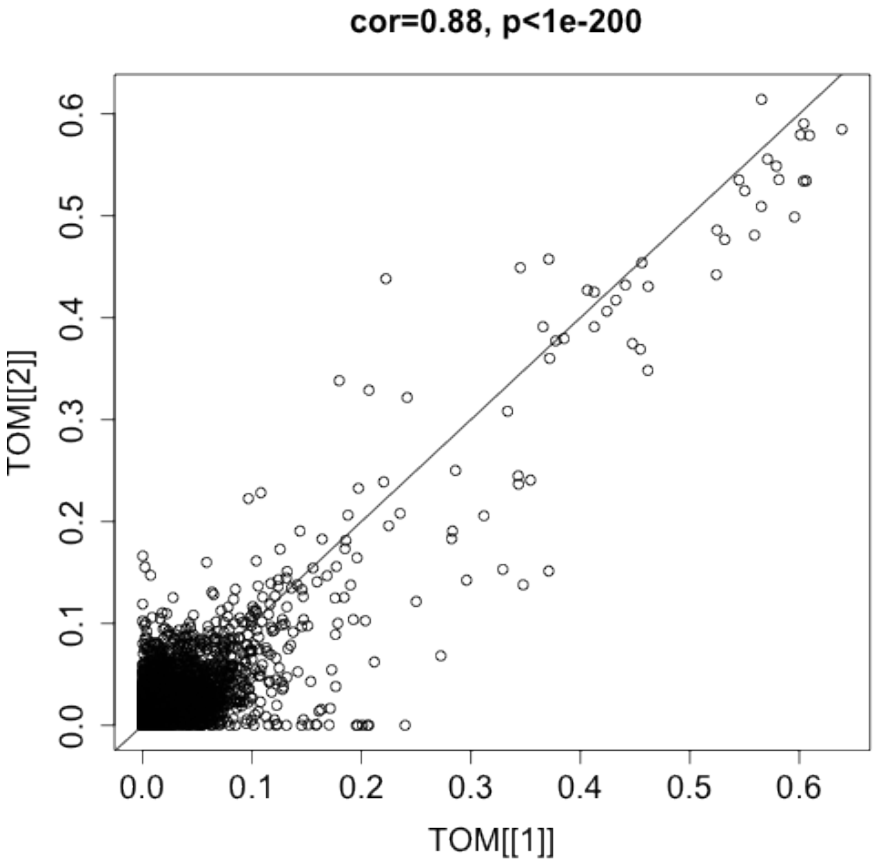

Figure 14: plot of chunk unnamed-chunk-40

```
consensusTOM = pmin(TOM[1, , ], TOM[2, , ])
consTreeManual = flashClust(as.dist(1 - consensusTOM), method = "average")
# Module identification using dynamic tree cut:
unmergedLabels = cutreeDynamic(dendro = consTreeManual, distM = 1 - consensusTOM,
    deepSplit = 2, cutHeight = 0.995, minClusterSize = 30, pamRespectsDendro = FALSE)
```

```
##  Detecting clusters...
```

```
unmergedColors = labels2colors(unmergedLabels)
```

The Dynamic Tree Cut may identify modules whose expression profiles are very similar. It may be prudent to merge such modules since their genes are highly co-expressed. To quantify co-expression similarity of entire modules, we calculate their eigengenes (MEs) and cluster them on their consensus correlation, that is the minimum correlation across the two sets:

```
# Calculate module eigengenes
unmergedMEs = multiSetMEs(multiExpr, colors = NULL, universalColors = unmergedColors)
```

```
##  multiSetMEs: Calculating module MEs.
##    Working on set 1 ...
##    Working on set 2 ...
```

```
# Calculate consensus dissimilarity of consensus module eigengenes
consMEDiss = consensusMEDissimilarity(unmergedMEs)
# Cluster consensus modules
consMETree = flashClust(as.dist(consMEDiss), method = "average")
# The merging can be performed automatically:
merge = mergeCloseModules(multiExpr, unmergedLabels, cutHeight = 0.25)
```

```
##  mergeCloseModules: Merging modules whose distance is less than 0.25
##    Calculating new MEs...
```

```
# resulting merged color
modLabConsManual0 = merge$colors

# Relabel the manual consensus modules so that their labels match those
# from the automatic analysis
modLabConsManual = matchLabels(modLabConsManual0, modLabCons)
# Agreement between consensus modules and females-only modules
mean(modLabConsManual == modLabCons)
```

```
## [1] 1
```

```r
# Convert the numeric labels to color labels
moduleColorsConsensusManual = labels2colors(modLabConsManual)
# Calculate consensus module eigengenes in each data set
consMEs = multiSetMEs(multiExpr, colors = NULL, universalColors = moduleColorsConsensusManua

##  multiSetMEs: Calculating module MEs.
##    Working on set 1 ...
##    Working on set 2 ...

# Plot the gene dendrogram again:
plotDendroAndColors(consTreeManual, moduleColorsConsensusManual, groupLabels = "Module",
    dendroLabels = FALSE, hang = 0.03, addGuide = TRUE, guideHang = 0.05)
```

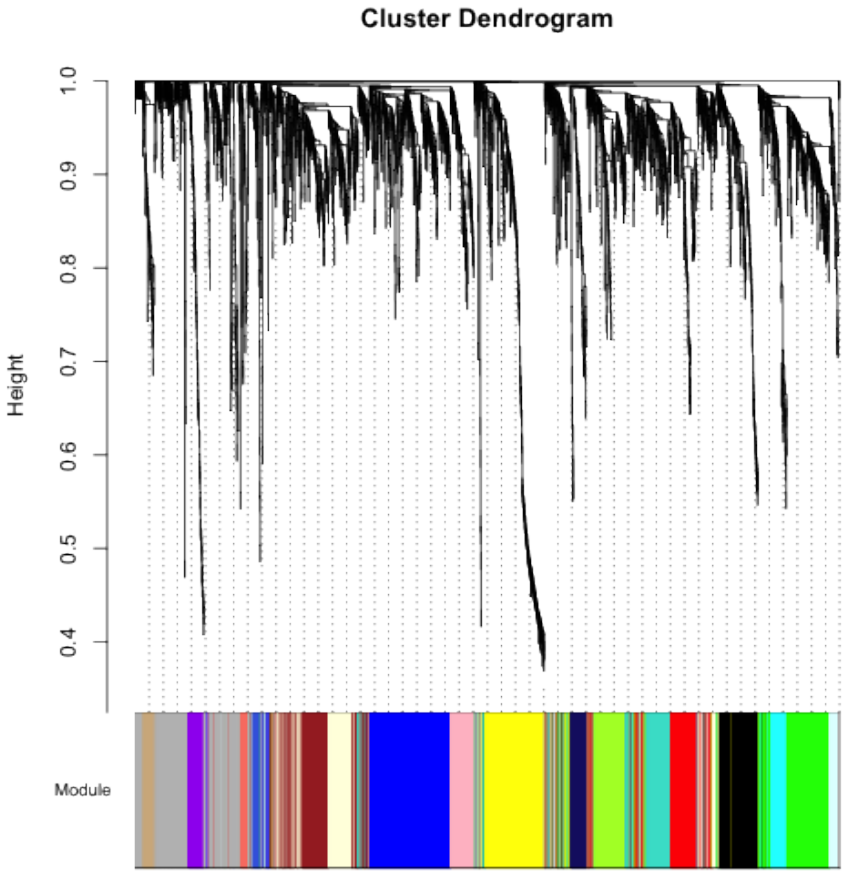

Figure 15: plot of chunk unnamed-chunk-43

Caption: The resulting plot is identical to that which was based on the automatic analysis implemented in the function blockwiseConsensusModules.

Exercises Exercise 1. Reverse the role of the female and male mouse liver data sets and re-analyze the data. In particular, carry out the following tasks. i) Construct a sample network based on the male mouse liver samples and identify potential outliers. ii) Use different module detection procedures for finding co-expression modules in the male liver samples. Hint: dynamic hybrid method: single block, 2 block, and manual module analysis. iii) Evaluate the agreement between the module detection methods. iv) Relate the module eigengenes to physiological traits in male mice. v) Evaluate module preservation of male modules in the female network. vi) Discuss whether you have found something interesting. Is it publishable?

Exercise 2 regarding visualizing a network. Use your favorite data set to define a weighted correlation network and modules (based on hierarchical clustering). Visualize the network and the modules using i) a connectivity (TOM) plot, ii) a classical multidimensional scaling plot, iii) a ViSANT plot, iv) Which plots or software packages allow one to visualize a weighted network? Which tools work only for an unweighted network? v) How can one visualize a weighted network with a tool that only visualizes unweighted networks? Answer: threshold the adjacency matrix.

Exercise 3 regarding systems genetic analysis with NEO. Recall that the analysis in section 12.3 focused on the blue module. Here you are asked to analyze the brown module instead. i) Show that the brown module correlates with body weight. ii) Determine which SNP(s) correlate with the brown module eigengene. Hint: Run cor(datSNP,MEbrown,use="p"). iii) Use the most significant SNP to determine whether SNP->MEbrown-> weight. iv) Which genes in the brown module show evidence of causally affecting body weight? Hint: use the SNP from iii) iv) How do the NEO results change if you use the second most significant SNP. v) How do the results change if you use the LEO.CPA score (which uses the two most significant SNPs)? Hint: Use the R code from section 11.8.2.

References Broman KW, Wu H, Sen S, Churchill GA (2003) R/qtl: QTL mapping in experimental crosses. Bioinformatics 19(7):889–890

Frohlich H, Speer N, Poustka A, Beiszbarth T (2007) GOSim – an R-package for computation of information theoretic GO similarities between terms and gene products. BMC Bioinform 8(1):166 1086

Fuller TF, Ghazalpour A, Aten JE, Drake T, Lusis AJ, Horvath S (2007) Weighted gene coexpression network analysis strategies applied to mouse weight. Mamm Genome 18(6–7):463–472

Ghazalpour A, Doss S, Zhang B, Plaisier C, Wang S, Schadt EE, Thomas A, Drake TA, Lusis AJ, Horvath S (2006) Integrating genetics and network analysis to characterize genes related to mouse weight. PloS Genet 2(2):8

Hu Z, Mellor J, Wu J, DeLisi C (2004) VisANT: An online visualization and analysis tool for biological interaction data. BMC Bioinform 5:17

Jansen RC, Nap JP (2001) Genetical genomics: The added value from segregation. Trends Genet 10(17):388

Langfelder P, Zhang B, Horvath S (2007) Defining clusters from a hierarchical cluster tree: The Dynamic Tree Cut library for R. Bioinformatics 24(5):719–720

Langfelder P, Luo R, Oldham MC, Horvath S (2011) Is my network module preserved and reproducible? Plos Comput Biol 7(1):e1001057

de Nooy W,Mrvar A, Batagelj V (2005) Exploratory social network analysis with pajek (structural analysis in the social sciences). Cambridge University Press, Cambridge

Presson AP, Sobel EM, Papp JC, Suarez CJ, Whistler T, Rajeevan MS, Vernon SD, Horvath S (2008) Integrated weighted gene co-expression network analysis with an application to chronic fatigue syndrome. BMC Syst Biol 2:95

Ravasz E, Somera AL, Mongru DA, Oltvai ZN, Barabasi AL (2002) Hierarchical organization of modularity in metabolic networks. Science 297(5586):1551–1555

Schadt EE, Monks SA, Drake TA, Lusis AJ, Che N, Colinayo V, Ruff TG, Milligan SB, Lamb JR, Cavet G, Linsley PS, Mao M, Stoughton RB, Friend SH (2003) Genetics of gene expression surveyed in maize, mouse and man. Nature 422:297–302

Shannon P, Markiel A, Ozier O, Baliga NS,Wang JT, Ramage D, Amin N, Schwikowski B, Ideker T (2003) Cytoscape: A software environment for integrated models of biomolecular interaction networks. Genome Res 13(11):2498–2504

Zhang B, Horvath S (2005) General framework for weighted gene coexpression analysis. Stat Appl Genet Mol Biol 4:17