Identifying Gene Co-Expression Modules in the Developing Human Brain

Course: MED263, "Bioinformatics Applications to Human Disease"
Preparer: Kevin Chau (kkchau@ucsd.edu) (https://github.com/kkchau/MED263)

Introduction

Gene co-expression networks are highly informative in the context of biological processes as well as identifying risk genes through a guilty-by-association framework. These co-expression networks are constructed such that nodes represent the elements of interest, the genes, and edges represent correlations between the expression patterns of those genes. A straighforward approach to the construction of these gene co-expression networks would consist of calculating pairwise correlations, e.g. Pearson correlations, and applying a hard threshold such that edges only exists between nodes that have a correlation that surpasses the threshold. However, this approach comes with the caveat of lost information. For example, if two genes exhibit a correlation very close to, but not meeting, an established threshold, that potential edge is lost.

In this practical, we will be using R, specifically the Weighted Gene Co-Expression Network Analysis (WGCNA) package, to identify and characterize gene co-expression modules from human brain developmental transcriptome expression data. We will be using publicly available gene expression data from the BrianSpan Atlas, creating the networks with WGCNA in R, and characterizing these modules with ENRICHR.
Note: For reference, the following code was executed in a Docker container allocated with 3 CPUs and 4096 MB of memory.

Set-Up

We will be using WGCNA for the actual network construction; data will be organized into *SummarizedExperiment* objects for ease of use. All of the following packages should already be installed if running from the corresponding Docker container.

```
setwd("/home/rstudio/work")
```

```
# install.packages(c('matrixStats', 'Hmisc', 'splines',
#                    'foreach', 'doParallel', 'fastcluster',
#                    'dynamicTreeCut', 'survival', 'viridisLite',
#                    'ggplot2', 'enrichR')
# source("https://bioconductor.org/biocLite.R")
# biocLite(c('GO.db', 'preprocessCore', 'imput', 'WGCNA', 'SummarizedExperiment'))
```

Libraries

```
library(WGCNA)
library(SummarizedExperiment)
library(enrichR)
library(ggplot2)
library(gridExtra)
```

```
### Optional:
# Enable parallel processing for computationally intensive tasks
# (soft thresholding, topological overlap, etc.)
# Go to your Docker settings (Settings -> Advanced) and check how many CPUs
# you would like to dedicate to the process
# Then, uncomment and run the following lines
library(doParallel)
allowWGCNAThreads()
```

```
## Allowing multi-threading with up to 3 threads.
```

Data

Initial Data Structuring

To begin the data analysis, we will first download and extract gene expression data from the BrainSpan Atlas.
NOTE: If running from the Docker container, the data files will already be downloaded

```
# url <- "http://www.brainspan.org/api/v2/well_known_file_download/267666525"
# utils::download.file(url, destfile="/home/rstudio/brainSpan.zip", mode='wb')
# utils::unzip("brainSpan.zip", exdir="/home/rstudio/brainSpan")
# file.remove("brainSpan.zip")
```


The downloaded files consist of a RPKM expression matrix, sample metadata, and row metadata, along with a readme file.

Question 1
How were expressions compiled (what were the tools used for alignment/quantification) and what was the reference?

Answer
Read alignment was performed with Tophat using Gencode v10 annotations; quantification was done with SAMtools and RSEQtools (this can be found in the whitepaper from https://brainspan.org)

Next, we want to package the relevant data and metadata together into a single SummarizedExperiment. This is done to protect the integrity of the data and safeguard value mappings, yielding a nice reference in case any downstream analysis goes wrong.

```
expr <- read.csv("/home/rstudio/brainSpan/expression_matrix.csv", header=FALSE)[, -1]
coldata <- read.csv("/home/rstudio/brainSpan/columns_metadata.csv")[, -1]
rowdata <- read.csv("/home/rstudio/brainSpan/rows_metadata.csv")[, -1]

row.names(expr) <- rowdata$ensembl_gene_id
colnames(expr) <- as.character(apply(coldata, 1,
                                     FUN=function(x) paste(x[["donor_name"]],
                                                           x[["structure_acronym"]],
                                                           sep="."
                                                           )
                                     )
                               )
se.expr <- SummarizedExperiment(assays=list(rpkm=as.matrix(expr)),
```

```
                                rowData=rowdata,
                                colData=coldata
                                )
se.expr
```

```
## class: SummarizedExperiment
## dim: 52376 524
## metadata(0):
## assays(1): rpkm
## rownames(52376): ENSG00000000003 ENSG00000000005 ...
##   ENSGR0000248421 ENSGR0000249358
## rowData names(4): gene_id ensembl_gene_id gene_symbol entrez_id
## colnames(524): H376.IIA.51.Ocx H376.IIA.51.M1C-S1C ...
##   H376.XI.56.STR H376.XI.56.S1C
## colData names(7): donor_id donor_name ... structure_acronym
##   structure_name
```

The se.expr SummarizedExperiment object is essentially a collection of matrices linked by the appropriate mappings; that is, the columns of the assays (in this case, the rpkm assay) correspond to the "colnames" of the SummarizedExperiment, which in turn have the attributes listed in the "colData" field. The same concept applies to the rownames and rowData.

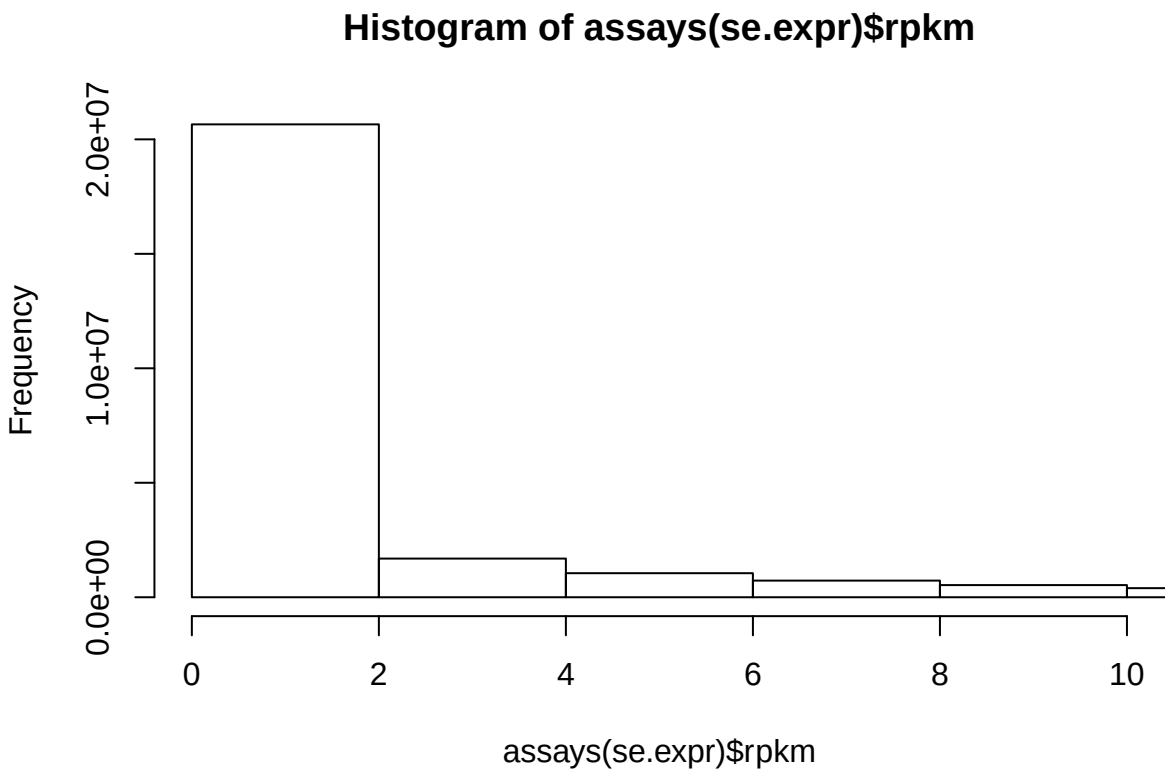
Methods and Results

Data filtering

Expression filtering

Now, we will filter this expression data for lowly expressed genes since these features are likely representative of noise in the dataset. Additionally, this will reduce the size of the data and make it more managable for the more computationally intensive steps to follow. We first plot the density plot of the expression distribution to identify a suitable cutoff value.

```
hist(assays(se.expr)$rpkm, xlim=c(0, 10), breaks=100000)
```



Given this plot we select an expression cutoff of 5 RPKM in at least 80% of samples (this is a very strict threshold, but it is necessary to reduce the data to a manageable size for the default Docker memory settings).

```
se.expr.filt <- se.expr[apply(assays(se.expr)$rpkm >= 5, 1, sum) >= 0.9*ncol(se.expr),]

dim(se.expr.filt)
```

```
## [1] 4805  524
```

```
object.size(se.expr.filt)
```

```
## 6692760 bytes
```

Sample clustering

We will now perform sample-wise clustering to identify any outlier samples, reducing the noise in the dataset as well as its size.
NOTE: Given memory constraints the following code is not executed

```
# sampleClust <- hclust(dist(assays(se.expr)$rpkm), method='average')
```

Question 2 Which five genes have the highest overall (average across samples) expression?

Answer ENSG00000252229(20443.862), ENSG00000252197(19443.306), ENSG00000240831(11947.972), ENSG00000243172(9417.093), ENSG00000239935(8261.913)

Soft thresholding

A stipulation of WGCNA network construction is that the data should satisfy the scale-free topology criteria. Scale-free networks are networks whose degree distributions (the number of neighbors per node) follows a power function, such that there are few nodes with a large number of neighbors whereas the rest of the nodes have few neighbors. This gives rise to "network hubs," and biologically this implies that there are fewer "vulnerable" genes such that their disruption results in the shutting down of a pathway, for example. This step consists of testing this topology on our data for different integer power levels. We want to select a power level that sufficiently satisfies the scale-free topology while still preserving connectivity.
NOTE: This will take a while to run. If possible, enable parallel processing with allowWGCNAThreads().

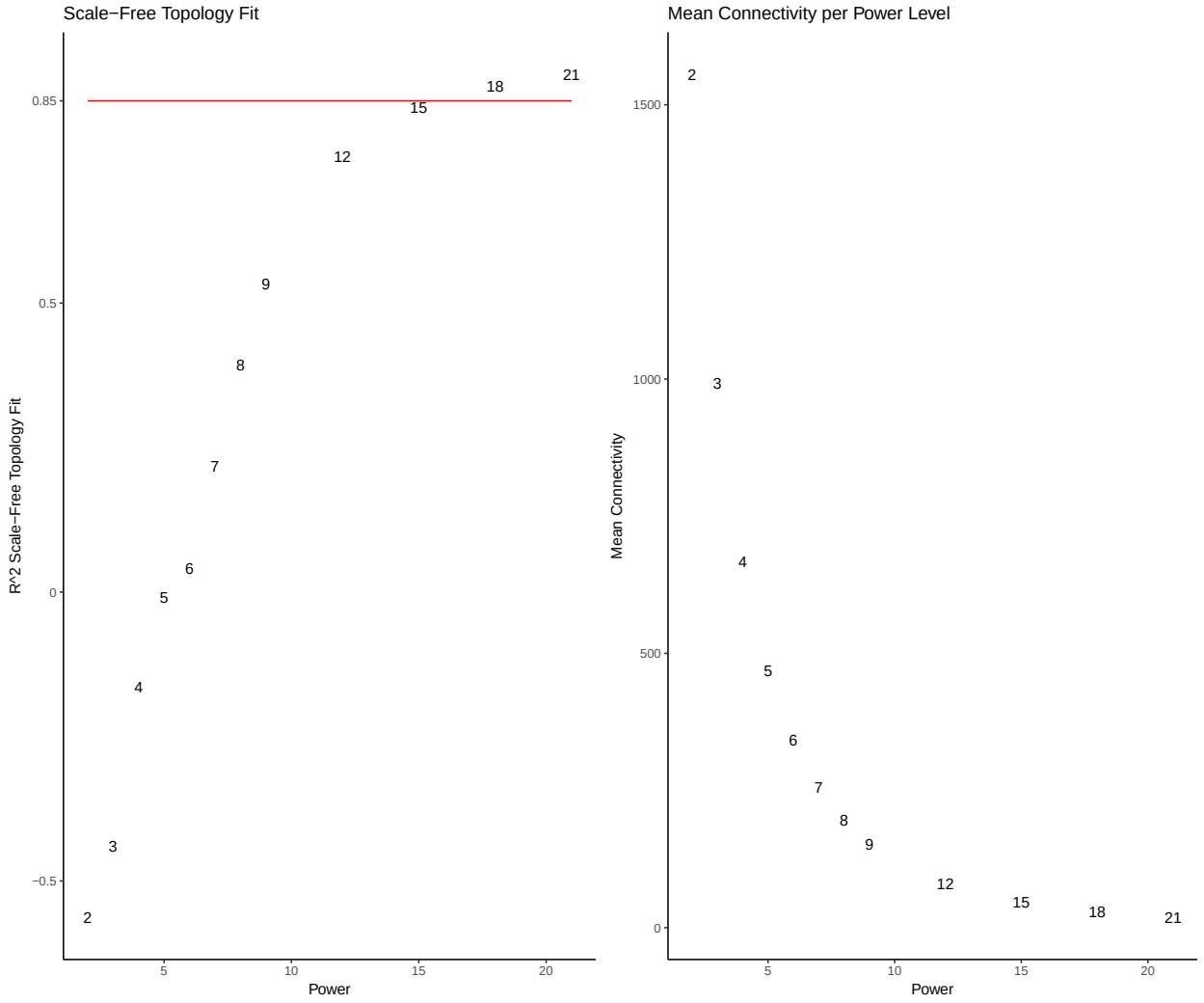
```
pwr <- c(seq(2, 8), seq(9, 21, by=3))
soft.thresh <- pickSoftThreshold(t(assays(se.expr.filt)$rpkm),
                                 powerVector=pwr,
                                 networkType="signed",
                                 verbose=5
                                 )
```

```
## pickSoftThreshold: will use block size 4805.
##  pickSoftThreshold: calculating connectivity for given powers...
##    ..working on genes 1 through 4805 of 4805
##    Power SFT.R.sq  slope truncated.R.sq mean.k. median.k. max.k.
## 1      2  0.56300  3.610          0.847  1560.0    1570.0   2000
## 2      3  0.44000  1.870          0.905   992.0     991.0   1440
## 3      4  0.16500  0.811          0.923   668.0     659.0   1110
## 4      5  0.00964  0.157          0.932   469.0     454.0    897
## 5      6  0.04160 -0.296          0.921   342.0     324.0    740
## 6      7  0.21700 -0.669          0.909   256.0     237.0    622
## 7      8  0.39400 -0.896          0.924   196.0     177.0    529
## 8      9  0.53400 -1.080          0.929   153.0     134.0    455
## 9     12  0.75300 -1.340          0.953    79.8      63.0    306
## 10    15  0.83900 -1.450          0.966    46.1      32.6    216
## 11    18  0.87600 -1.480          0.958    28.6      18.2    159
## 12    21  0.89600 -1.500          0.957    18.7      10.6    120
```

We can then plot the R^2 fit to the scale-free topology of the different power levels as well as the mean connectivity.

```
sftplt <- (ggplot(soft.thresh$fitIndices,
                  aes(x=soft.thresh$fitIndices$Power,
                      y=-sign(soft.thresh$fitIndices$slope)*soft.thresh$fitIndices$SFT.R.sq)
                  )
           + geom_text(label=soft.thresh$fitIndices$Power)
           + scale_y_continuous(breaks=c(-0.5, 0, 0.5, 0.85, 1),
                                labels=c('-0.5', '0', '0.5', '0.85', '1')
                                )
           + geom_line(y=0.85, colour='red')
           + labs(x="Power",
                  y="R^2 Scale-Free Topology Fit",
                  title="Scale-Free Topology Fit"
                  )
           + theme(panel.grid.major=element_blank(),
                   panel.grid.minor=element_blank(),
                   panel.background=element_blank(),
                   axis.line=element_line())
           )
```

```
conplt <- (ggplot(soft.thresh$fitIndices,
                  aes(x=soft.thresh$fitIndices$Power,
                      y=soft.thresh$fitIndices$mean.k)
                  )
           + geom_text(label=soft.thresh$fitIndices$Power)
           + labs(x="Power",
                  y="Mean Connectivity",
                  title="Mean Connectivity per Power Level"
                  )
           + theme(panel.grid.major=element_blank(),
                   panel.grid.minor=element_blank(),
                   panel.background=element_blank(),
                   axis.line=element_line())
           )
grid.arrange(sftplt, conplt, ncol=2)
```



Question 3 What should we select as the power level for this data set?

Answer Some integer around 15. This is the point where we have sufficient correlation with the scale-free topology criterion. Choosing a greater value leads to decreased connectivity. This is also the value reported in *soft.thresh$powerEstimate*.

Adjacencies and the Topological Overlap Matrix

WGCNA constructs networks by first creating a correlation adjacency matrix, where pairwise correlations between each feature (gene) is calculated. These correlations are then scaled by the power level reported by the soft thresholding function. A topological overlap matrix (TOM) is then the result of taking the information provided by the adjacency matrix, essentially correlation values, and incorporating topological similarities, like neighbors and distances between nodes.

```
adj <- adjacency(t(assays(se.expr.filt)$rpkm), power=soft.thresh$powerEstimate)
tom <- TOMsimilarity(adj)
```

```
## ..connectivity..
## ..matrix multiplication (system BLAS)..
## ..normalization..
## ..done.
```

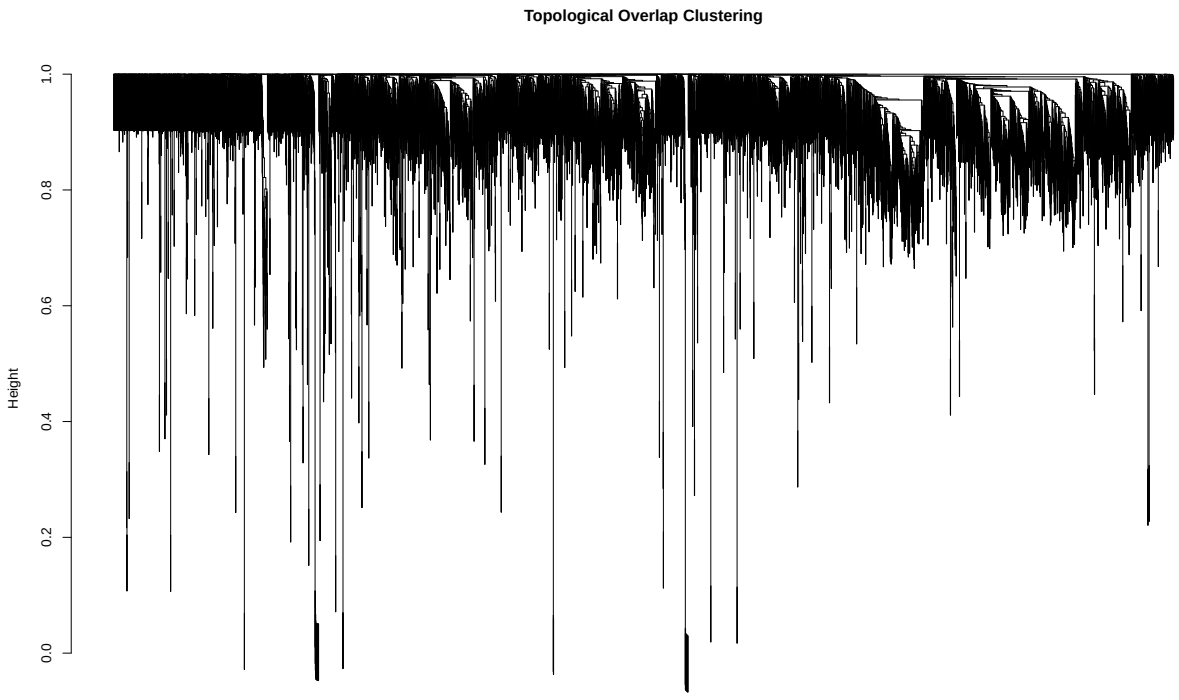
```
distom <- 1 - tom
```

Clustering of the TOM

We can perform hierarchical clustering on this TOM to cluster genes together

```
geneTree <- hclust(as.dist(distom), method='average')
```

```
plot(geneTree, xlab="", sub="", main="Topological Overlap Clustering", labels=FALSE)
```



Using *cutreeDynamic* to dynamically cut the dendrogram, we can perform a preliminary identification of gene modules.

```
modSize <- 20   # minimum module size
dynMods <- cutreeDynamic(dendro=geneTree, distM=distom, minClusterSize=modSize)
```

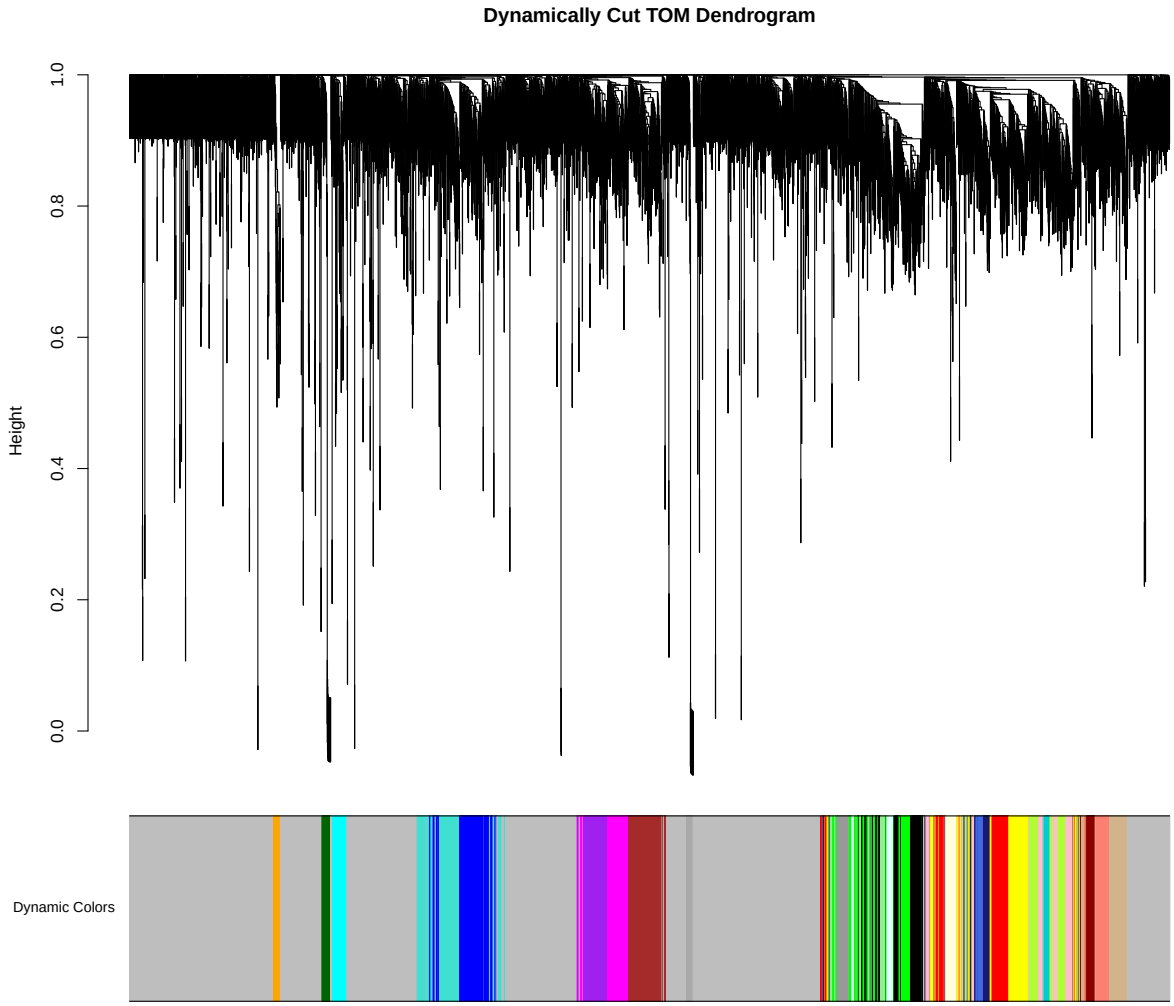
```
##  ..cutHeight not given, setting it to 0.998  ===>  99% of the (truncated) height range in dendro.
##  ..done.
```

```
table(dynMods)
```

```
## dynMods
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14
## 2457  190  179  171  170  149  146  135  132  117  113  111  101   75   67
##   15   16   17   18   19   20   21   22   23   24   25
##   59   54   51   51   49   49   43   40   35   31   30
```

This result shows that we have identified 25 modules (module 0 represents genes that cannot be classified into another module). We can plot module assignment under the dendrogram.

```
dynCol <- labels2colors(dynMods)
plotDendroAndColors(dendro=geneTree, colors=dynCol, dendroLabels=FALSE,
                    main="Dynamically Cut TOM Dendrogram", groupLabels="Dynamic Colors"
                    )
```

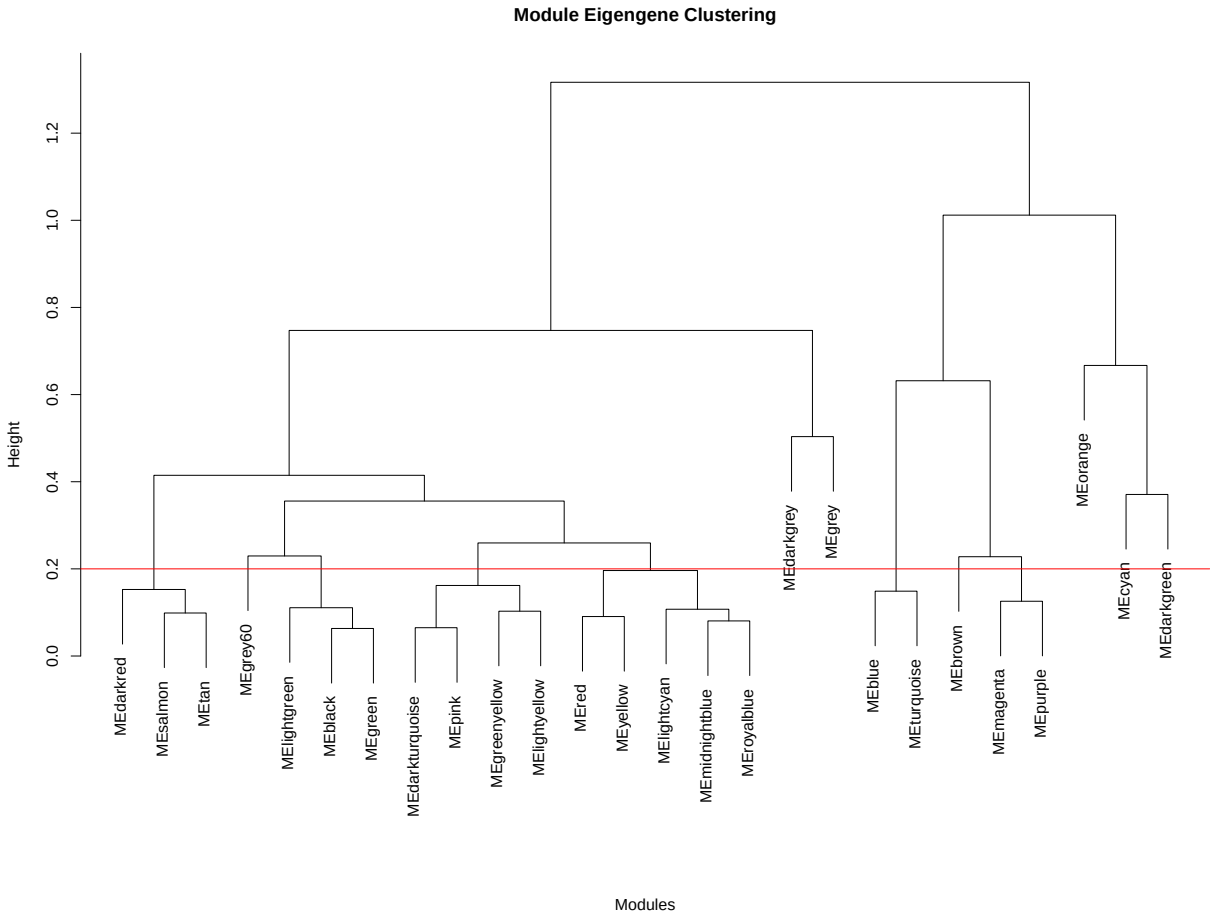


Question 4 What is a likely cause of such a large grey module?

Answer Due to such strict thresholding, many genes are now left with insufficient co-expression neighbors and thus do not make the clustering cutoff.

We can further refine the module identification by clustering highly correlated modules together. This is done based on the "module eigengenes," or representative vectors of each module that incorporates an average expression pattern of the module members. We will use a correlation of 0.8 as the merging threshold.

```
mergeThresh <- 1 - 0.8
MEs <- moduleEigengenes(t(assays(se.expr.filt)$rpkm), colors=dynCol)$eigengenes
disMEs <- 1 - cor(MEs)
METree <- hclust(as.dist(disMEs), method='average')
plot(METree, main="Module Eigengene Clustering", xlab="Modules", sub="")
abline(h=mergeThresh, col='red')
```




```
mergeEigen <- mergeCloseModules(t(assays(se.expr.filt)$rpkm), dynCol, cutHeight=mergeThresh)
```

```
##  mergeCloseModules: Merging modules whose distance is less than 0.2
##    Calculating new MEs...
```

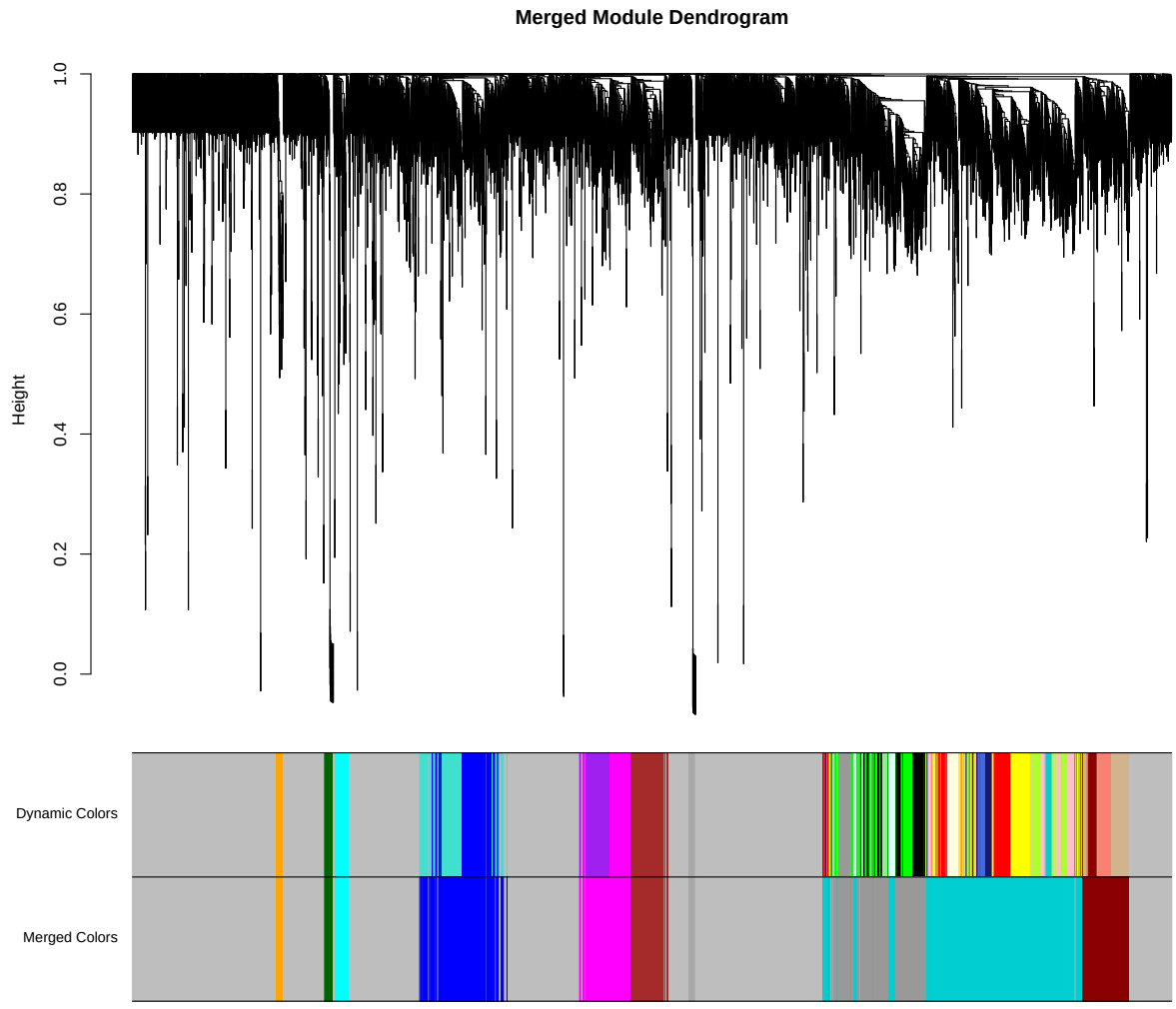
```
mergedCol <- mergeEigen$colors
mergedMEs <- mergeEigen$newMEs
table(mergedCol)
```

```
## mergedCol
```

```
##          blue         brown          cyan     darkgreen      darkgrey
##           369           171            67            40            31
##       darkred darkturquoise          grey        grey60       magenta
##           219           805          2457           386           230
##        orange
##            30
```

We can now plot the result of the merge

```
plotDendroAndColors(dendro=geneTree, colors=cbind(dynCol, mergedCol),
                    groupLabels=c("Dynamic Colors", "Merged Colors"),
                    dendroLabels=FALSE, main="Merged Module Dendrogram"
                    )
```



We now add the module assignments to our *se.expr.filt* object for safekeeping.

```
rowData(se.expr.filt)$module <- mergedCol
```

We can also export each module and their members into separate text files.

```
# Create directory for modules
dir.create("/home/rstudio/work/modules", showWarnings=F)
```

```
for (clr in unique(rowData(se.expr.filt)$module)) {
    fname <- paste0("/home/rstudio/work/modules/", clr, ".module")
    write.table(row.names(se.expr.filt)[rowData(se.expr.filt)$module==clr],
                file=fname, sep='\n', quote=F, row.names=F, col.names=F)
}
```

Functional Enrichment Analysis

Functional enrichment analysis will be done with the *enrichR* package. Here, we will analyze each module and determine what are the overarching "theme" of the module; that is, if there an overrepresentation of some kind of functionality exhibited by each particular set of genes found in each module.

EnrichR is a tool that takes a list of gene symbols, compares that list to a specified background database, and outputs the enrichment. In our case, we will be using the 'GO_Biological_Process_2017' database to query for Gene Ontology terms related to biological processes.

```
# Create enrichment directory
dir.create("/home/rstudio/work/enrichment", showWarnings=F)
db <- 'GO_Biological_Process_2017'
# We can ignore the grey module since those genes are just unassigned nodes
for (fmod in list.files("/home/rstudio/work/modules",
                        full.names=TRUE,
                        pattern="*.module")
     ) {
    if (fmod == "/home/rstudio/work/modules/grey.module") { next }
    # Get module color
    ofilename <- paste("/home/rstudio/work/enrichment/",
                       unlist(strsplit(fmod, "/"))[[6]],
                       ".enrich",
                       sep=""
                       )
    module.members <- scan(fmod, what=character())
    members.as.symbols <- rowData(se.expr.filt)$gene_symbol[rowData(se.expr.filt)$ensembl_gene_id %in% m
    this.enrichment <- enrichr(as.character(members.as.symbols), databases=db)
    printEnrich(this.enrichment, columns=c(1, 2, 3, 4, 7, 9), file=ofilename)
}
```

```
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2017... Done.
## Parsing results... Done.
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2017... Done.
## Parsing results... Done.
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2017... Done.
## Parsing results... Done.
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2017... Done.
## Parsing results... Done.
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2017... Done.
## Parsing results... Done.
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2017... Done.
```

```
## Parsing results... Done.
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2017... Done.
## Parsing results... Done.
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2017... Done.
## Parsing results... Done.
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2017... Done.
## Parsing results... Done.
## Uploading data to Enrichr... Done.
##   Querying GO_Biological_Process_2017... Done.
## Parsing results... Done.
```

Question 5 What does each column represent? What is the top GO term in the darkgreen module? What is its adjusted P-value?

Answer Columns can be found by exploring the *this.experiment$GO_Biological_Process_2017* matrix; they are 'Term', 'Overlap', 'P.value', 'Adjusted.P.value', 'Z.score', and 'Genes'. The top GO term in the darkgreen module is nervous system development with an adjusted p-value of 2.227390e-24.

```
sessionInfo()
```

```
## R version 3.4.3 (2017-11-30)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Debian GNU/Linux 9 (stretch)
##
## Matrix products: default
## BLAS: /usr/lib/openblas-base/libblas.so.3
## LAPACK: /usr/lib/libopenblasp-r0.2.19.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=C
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] doParallel_1.0.11          iterators_1.0.9
##  [3] foreach_1.4.4              gridExtra_2.3
##  [5] ggplot2_2.2.1              enrichR_1.0
##  [7] SummarizedExperiment_1.8.1 DelayedArray_0.4.1
##  [9] matrixStats_0.53.1         Biobase_2.38.0
## [11] GenomicRanges_1.30.3       GenomeInfoDb_1.14.0
## [13] IRanges_2.12.0             S4Vectors_0.16.0
## [15] BiocGenerics_0.24.0        WGCNA_1.63
## [17] fastcluster_1.1.24         dynamicTreeCut_1.63-1
##
## loaded via a namespace (and not attached):
##  [1] httr_1.3.1             jsonlite_1.5           bit64_0.9-7
```

```
##  [4] splines_3.4.3          Formula_1.2-2          latticeExtra_0.6-28
##  [7] blob_1.1.0             fit.models_0.5-14      GenomeInfoDbData_1.0.0
## [10] yaml_2.1.16            robustbase_0.92-8      impute_1.52.0
## [13] pillar_1.2.0           RSQLite_2.0            backports_1.1.2
## [16] lattice_0.20-35        digest_0.6.15          RColorBrewer_1.1-2
## [19] XVector_0.18.0         checkmate_1.8.5        colorspace_1.3-2
## [22] htmltools_0.3.6        preprocessCore_1.40.0  Matrix_1.2-12
## [25] plyr_1.8.4             pcaPP_1.9-73           pkgconfig_2.0.1
## [28] zlibbioc_1.24.0        GO.db_3.5.0            mvtnorm_1.0-7
## [31] scales_0.5.0           htmlTable_1.11.2       tibble_1.4.2
## [34] nnet_7.3-12            lazyeval_0.2.1         survival_2.41-3
## [37] magrittr_1.5           evaluate_0.10.1        memoise_1.1.0
## [40] MASS_7.3-49            foreign_0.8-69         rsconnect_0.8.5
## [43] tools_3.4.3            data.table_1.10.4-3    stringr_1.3.0
## [46] munsell_0.4.3          cluster_2.0.6          AnnotationDbi_1.40.0
## [49] compiler_3.4.3         rlang_0.2.0            grid_3.4.3
## [52] RCurl_1.95-4.10        rstudioapi_0.7         rjson_0.2.15
## [55] htmlwidgets_1.0        robust_0.4-18          rmarkdown_1.8
## [58] labeling_0.3           bitops_1.0-6           base64enc_0.1-3
## [61] gtable_0.2.0           codetools_0.2-15       curl_3.1
## [64] DBI_0.7                R6_2.2.2               rrcov_1.4-3
## [67] knitr_1.20             bit_1.1-12             rprojroot_1.3-2
## [70] Hmisc_4.1-1            stringi_1.1.6          Rcpp_0.12.15
## [73] rpart_4.1-13           acepack_1.4.1          DEoptimR_1.0-8
```

References

1. Allen Institute for Brain Science (2018). *BrainSpan Atlas of the Developing Human Brain.* RNA-seq Gencode v10 summarized to genes. Available from https://brainspan.org
2. Chen, E. Y., Tan, C. M., Kou, Y., Duan, Q., Wang, Z., Meirelles, G. V., ... Ma'ayan, A. (2013). *Enrichr: interactive and collaborative HTML5 gene list enrichment analysis tool.* BMC Bioinformatics, 14, 128. http://doi.org/10.1186/1471-2105-14-128
3. Langfelder, P., & Horvath, S. (2008). *WGCNA: an R package for weighted correlation network analysis.* BMC Bioinformatics, 9, 559. http://doi.org/10.1186/1471-2105-9-559
4. Kuleshov, M. V., Jones, M. R., Rouillard, A. D., Fernandez, N. F., Duan, Q., Wang, Z., ... Ma'ayan, A. (2016). *Enrichr: a comprehensive gene set enrichment analysis web server 2016 update.* Nucleic Acids Research, 44(Web Server issue), W90–W97. http://doi.org/10.1093/nar/gkw377