Bladder cohort - R Notebook

Adam Cankaya
acankaya2017@fau.edu
Based on tutorial found at https://www.costalab.org/wp-content/uploads/2020/11/R_class_D3.htm

First install BiocManager, edgeR, and TCGAbiolinks

```
ptm <- proc.time()  # start the timer

if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

BiocManager::install("edgeR")
```

```
## Bioconductor version 3.19 (BiocManager 1.30.23), R 4.4.0 (2024-04-24 ucrt)

## Warning: package(s) not installed when version(s) same as or greater than current; use
##   'force = TRUE' to re-install: 'edgeR'
```

```
BiocManager::install("TCGAbiolinks")
```

```
## Bioconductor version 3.19 (BiocManager 1.30.23), R 4.4.0 (2024-04-24 ucrt)

## Warning: package(s) not installed when version(s) same as or greater than current; use
##   'force = TRUE' to re-install: 'TCGAbiolinks'
```

```
BiocManager::install("genefilter")
```

```
## Bioconductor version 3.19 (BiocManager 1.30.23), R 4.4.0 (2024-04-24 ucrt)

## Warning: package(s) not installed when version(s) same as or greater than current; use
##   'force = TRUE' to re-install: 'genefilter'
```

```
chunk1_time = (proc.time() - ptm)[3] # stop the timer
```

TODO - rename steps to match steps from slides Step 1 - Load packages, download data from TCGA, and prepare it for DEGList

```r
ptm <- proc.time()  # start the timer

library("TCGAbiolinks")
library("limma")
library("edgeR")
library("glmnet")
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```r
library("factoextra")
```

```
## Loading required package: ggplot2
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
library("FactoMineR")
library("caret")
```

```
## Loading required package: lattice
```

```r
library("SummarizedExperiment")
```

```
## Loading required package: MatrixGenerics
```

```
## Loading required package: matrixStats
```

```
##
## Attaching package: 'MatrixGenerics'
```

```
## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
##     rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars
```

```
## Loading required package: GenomicRanges
```

```
## Loading required package: stats4

## Loading required package: BiocGenerics

##
## Attaching package: 'BiocGenerics'

## The following object is masked from 'package:limma':
##
##     plotMA

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, table,
##     tapply, union, unique, unsplit, which.max, which.min

## Loading required package: S4Vectors

##
## Attaching package: 'S4Vectors'

## The following objects are masked from 'package:Matrix':
##
##     expand, unname

## The following object is masked from 'package:utils':
##
##     findMatches

## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname

## Loading required package: IRanges

##
## Attaching package: 'IRanges'

## The following object is masked from 'package:grDevices':
##
##     windows

## Loading required package: GenomeInfoDb
```

```
## Loading required package: Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

##
## Attaching package: 'Biobase'

## The following object is masked from 'package:MatrixGenerics':
##
##     rowMedians

## The following objects are masked from 'package:matrixStats':
##
##     anyMissing, rowMedians
```

```
library("gplots")
```

```
##
## Attaching package: 'gplots'

## The following object is masked from 'package:IRanges':
##
##     space

## The following object is masked from 'package:S4Vectors':
##
##     space

## The following object is masked from 'package:stats':
##
##     lowess
```

```
library("survival")
```

```
##
## Attaching package: 'survival'

## The following object is masked from 'package:caret':
##
##     cluster
```

```
library("survminer")
```

```
## Loading required package: ggpubr
```

```
##
## Attaching package: 'survminer'

## The following object is masked from 'package:survival':
##
##     myeloma
```

```
library("RColorBrewer")
library("gProfileR")
library("genefilter")
```

```
##
## Attaching package: 'genefilter'

## The following objects are masked from 'package:MatrixGenerics':
##
##     rowSds, rowVars

## The following objects are masked from 'package:matrixStats':
##
##     rowSds, rowVars
```

```
setwd('C:/Adam/R/')  # make sure it already exists

ptm <- proc.time()  # start the timer

# Before we perform a GDC query let's look at the TCGA-BLCA data
# As of June 2024 we should see a case count of 412
TCGAbiolinks:::getProjectSummary("TCGA-BLCA")
```

```
## $file_count
## [1] 23394
##
## $data_categories
##    file_count case_count                data_category
## 1        6729        412  Simple Nucleotide Variation
## 2        4285        412              Sequencing Reads
## 3        1760        412                   Biospecimen
## 4         994        412                      Clinical
## 5        4478        412        Copy Number Variation
## 6        1736        412      Transcriptome Profiling
## 7        1320        412               DNA Methylation
## 8         343        343           Proteome Profiling
## 9          26         12 Somatic Structural Variation
## 10       1723        406         Structural Variation
##
## $case_count
## [1] 412
##
## $file_size
## [1] 4.082979e+14
```

```r
# Download TCGA-BLCA data from GDC
# We want the complete RNA sequencing and raw gene count data
# So we run a query of the Transcriptome Profiling category and RNA-Seq experimental type
# We use the STAR - Counts workflow type because it contains the raw gene counts we need
# We ignore other sample types besides tumor and normal
# The original paper by Wang uses the HTSeq-counts workflow, but this is a legacy version of
#  the new STAR - COUNTS workflow type
query_TCGA = GDCquery(
  project = "TCGA-BLCA",
  data.category = "Transcriptome Profiling",
  data.type="Gene Expression Quantification",
  experimental.strategy = "RNA-Seq",
  workflow.type = "STAR - Counts",
  sample.type = c("Primary Tumor", "Solid Tissue Normal"))
```

```
## ------------------------------------

## o GDCquery: Searching in GDC database

## ------------------------------------

## Genome of reference: hg38

## ------------------------------------------

## oo Accessing GDC. This might take a while...

## ------------------------------------------

## ooo Project: TCGA-BLCA

## --------------------

## oo Filtering results

## --------------------

## ooo By experimental.strategy

## ooo By data.type

## ooo By workflow.type

## ooo By sample.type

## ----------------

## oo Checking data
```

```
## ----------------

## ooo Checking if there are duplicated cases

## ooo Checking if there are results for the query

## ------------------

## o Preparing output

## ------------------
```

```
# Run the query and format it as a table
# The results are a table with 431 rows (because some patients have multiple cases each)
# There are 29 columns with meta data about each case such as sample_type (tumor vs normal)
lihc_res = getResults(query_TCGA)

# We can create a summary table shows there are 412 tumor and 19 normal (412+19=431)
summary(factor(lihc_res$sample_type))
```

```
##       Primary Tumor Solid Tissue Normal
##                 412                  19
```

```
# Go ahead and download all the data from GDC to our working directory
GDCdownload(query = query_TCGA)
```

```
## Downloading data for project TCGA-BLCA

## Of the 431 files for download 431 already exist.

## All samples have been already downloaded
```

```
# Now load the RNA-Seq data from the files into R workspace
tcga_data = GDCprepare(query_TCGA)
```

```
## |                                                  |   0%                    |

## Starting to add information to samples

##  => Add clinical information to samples

##  => Adding TCGA molecular information from marker papers

##  => Information will have prefix 'paper_'

## blca subtype information from:doi:10.1016/j.cell.2017.09.007
```

```
## Available assays in SummarizedExperiment :
##   => unstranded
##   => stranded_first
##   => stranded_second
##   => tpm_unstrand
##   => fpkm_unstrand
##   => fpkm_uq_unstrand
```

```
# This data object has 60660 rows and 431 columns
# This indicates there are 60660 different genes found throughout all the cases
# The object contains both clincal and expression data
dim(tcga_data)
```

```
## [1] 60660   431
```

```
# We can access the data in the object like this which verifies 412 tumor and 19 normal
table(tcga_data@colData$definition)
```

```
##
## Primary solid Tumor Solid Tissue Normal
##                 412                  19
```

```
# Or see the gender data of 117 female and 314 male
table(tcga_data@colData$gender)
```

```
##
## female   male
##    117    314
```

```
# let's look at the various names of the first 6 genes...
head(rowData(tcga_data))
```

```
## DataFrame with 6 rows and 10 columns
##                       source     type     score     phase            gene_id
##                     <factor> <factor> <numeric> <integer>        <character>
## ENSG00000000003.15   HAVANA     gene        NA        NA ENSG00000000003.15
## ENSG00000000005.6    HAVANA     gene        NA        NA  ENSG00000000005.6
## ENSG00000000419.13   HAVANA     gene        NA        NA ENSG00000000419.13
## ENSG00000000457.14   HAVANA     gene        NA        NA ENSG00000000457.14
## ENSG00000000460.17   HAVANA     gene        NA        NA ENSG00000000460.17
## ENSG00000000938.13   HAVANA     gene        NA        NA ENSG00000000938.13
##                         gene_type   gene_name       level     hgnc_id
##                       <character> <character> <character> <character>
## ENSG00000000003.15 protein_coding      TSPAN6           2  HGNC:11858
## ENSG00000000005.6  protein_coding        TNMD           2  HGNC:17757
## ENSG00000000419.13 protein_coding        DPM1           2   HGNC:3005
## ENSG00000000457.14 protein_coding       SCYL3           2  HGNC:19285
## ENSG00000000460.17 protein_coding    C1orf112           2  HGNC:25565
## ENSG00000000938.13 protein_coding         FGR           2   HGNC:3697
##                            havana_gene
##                            <character>
```

```
## ENSG00000000003.15 OTTHUMG00000022002.2
## ENSG00000000005.6  OTTHUMG00000022001.2
## ENSG00000000419.13 OTTHUMG00000032742.2
## ENSG00000000457.14 OTTHUMG00000035941.6
## ENSG00000000460.17 OTTHUMG00000035821.9
## ENSG00000000938.13 OTTHUMG00000003516.3
```

```
# To preview the raw gene counts let's look at the expression levels of the first
#  6 genes in the first 3 cases...
rownames = values(tcga_data)$gene_name[1:6]
firs6genes = head(assay(tcga_data)[,1:3])
rownames(firs6genes) = rownames
colnames(firs6genes) = c("Case 1", "Case 2", "Case 3")
firs6genes
```

```
##          Case 1 Case 2 Case 3
## TSPAN6     3679  28986    951
## TNMD          0     21      1
## DPM1       4190   2917   2976
## SCYL3       850   1910    705
## C1orf112   1196   1495    655
## FGR         353    905   2282
```

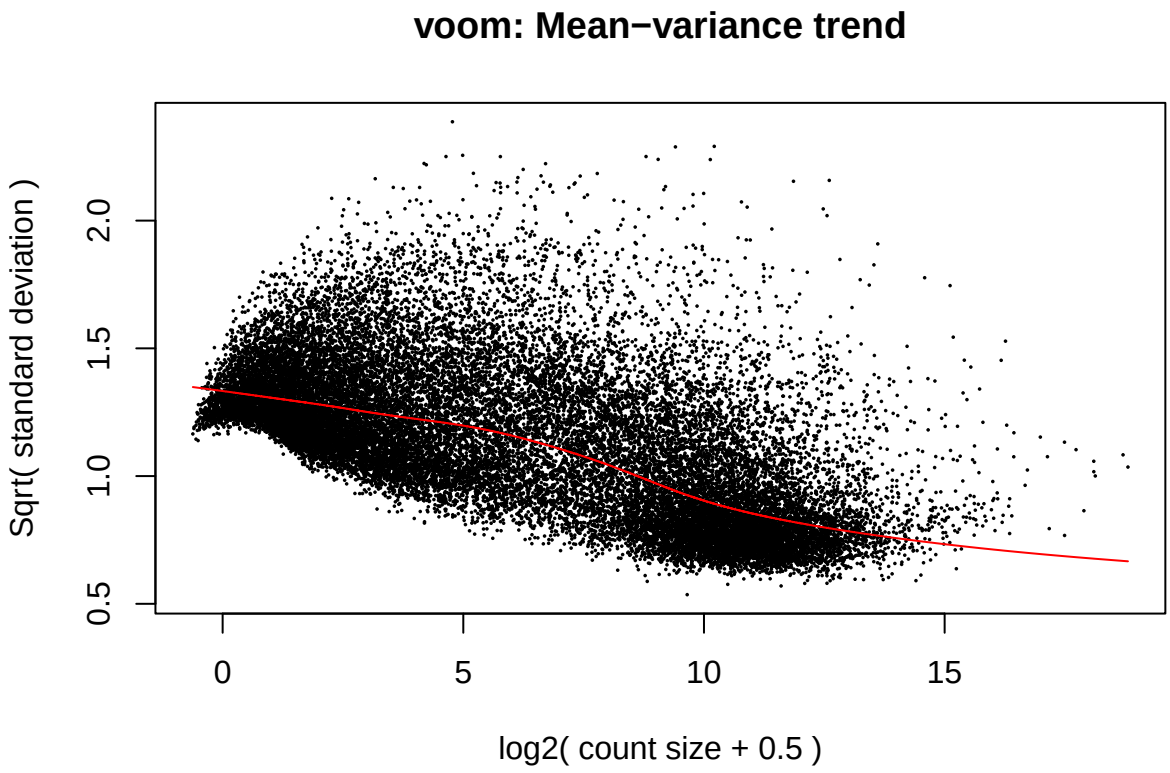
```
chunk2_time = (proc.time() - ptm)[3] # stop the timer
```

Step 2 - Generate DGEList, filter low counts, and normalize data

```
# Before we can perform DEG analysis we need to normalize the data
# Let's create a limma pipeline to do this...
# The pipeline function will take in three input parameters:
#   tcga_data - the data object we created in Step 1
#   condition_variable - the variable by which we will group patients (tumor vs normal)
#   reference_group - indicates which of the condition variable
#    values is the reference group (no tumors)
# The pipeline will return a list of three objects:
#   voom - the TMM normalized data returned by running voom
#   eBayes - the fitted model returned by running eBayes
#   topTable - a simple table which contains the top 1000 differentially expressed genes
#    sorted by p.value
limma_pipeline = function(
  tcga_data,
  condition_variable,
  reference_group=NULL) {

    # Create a design matrix
    # The factor is the category classifier for the data (tumor vs normal)
    #  limma requires it to be a factor object
    design_factor = colData(tcga_data)[, condition_variable, drop=T] # definition
    group = factor(design_factor) # Solid Normal Tissue

    # otherwise just pick the first class as the reference class
```

```
    if (!is.null(reference_group)) {
      group = relevel(group, ref=reference_group)
    }

    # make the design matrix
    design = model.matrix(~ group)

    # generate the DGEList object using the input...
    #  counts is the raw gene counts (numericla matrix - rows as genes, columns as cases)
    #  samples is the clinical data (data frame)
    #  genes is the annotation information (data frame - gene id and names)
    # the DGEList object returned is a transformed version of tcga_data
    dge = DGEList(counts=assay(tcga_data),
                  samples=colData(tcga_data),
                  genes=as.data.frame(rowData(tcga_data)))

    # filtering - by default genes with less than 10 counts per million reads are removed
    # after filtering we have 28087 genes remaining
    # no need to filter further by logfc or adjusted p-value because all
    #  entries already meet the cutoff criteria
    keep = filterByExpr(dge,design) # genes which meet are left after filtering
    dge = dge[keep,,keep.lib.sizes=FALSE] # filter the DGEList object, only keep the genes we want
    rm(keep) # remove this object from memory because we are done with it

    # TODO do we need rpkm() filtering?

    # Normalization (TMM followed by voom)
    # normalizing - minimize batch effects and variation with the TMM normalization
    # TMM - trimmed mean of M-values
    # use the voom method to convert the data to have a similar variance as arrays
    dge = calcNormFactors(dge)
    v = voom(dge, design, plot=TRUE)

    # Fit model to data given design
    #  fits a series of linear models, one to each probe
    #  then pass it to eBayes to rank the differential expression
    fit = lmFit(v, design)
    fit = eBayes(fit)

    # Save top genes
    topGenes = topTable(fit, coef=ncol(design), number=1000, sort.by="p")

    return(
      list(
        voomObj=v, # normalized data
        fit=fit, # fitted model and statistics
        topGenes=topGenes # the 1000 most differentially expressed genes
      )
    )
}

# Run the pipeline on the tcga_data from step 1 and normal tissue as the reference
#  "definition" is the column name for the tissue type (tumor vs normal)
```

```
#  "Solid Tissue Normal" is our baseline/control/reference class value
# The limma_res object returned is a list of 3 objects - voomObj, fit, topGenes
ptm <- proc.time()  # start the timer
limma_res = limma_pipeline(
  tcga_data=tcga_data,
  condition_variable="definition",
  reference_group="Solid Tissue Normal"
)
```

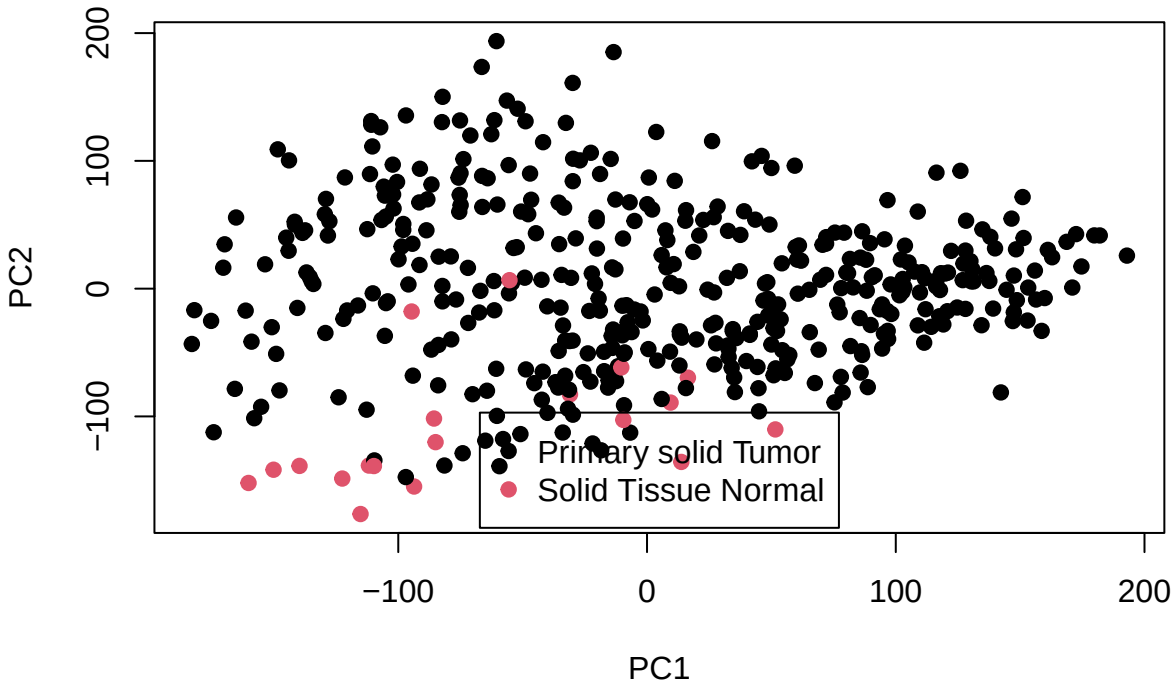



```
chunk3_time = (proc.time() - ptm)[3] # stop the timer
```

Step 3 - Visualize

```
ptm <- proc.time()  # start the timer

# make a function to generate a scatter plot to show a separation of tumor vs normal points
plot_PCA = function(voomObj, condition_variable){
  # create a factor
  group = factor(voomObj$targets[, condition_variable])
  # perform a principal component analysis
  pca = prcomp(t(voomObj$E))
  # Take PC1 and PC2 for the plot
  plot(pca$x[,1:2],col=group, pch=19)
```

```
  # include a legend for points
  legend("bottom", inset=.01, levels(group), pch=19, col=1:length(levels(group)))
  return(pca)
}

# call the plot function with the voom object and the defintion column
res_pca = plot_PCA(limma_res$voomObj, "definition")
```




```
# create a volcano plot
x = limma_res$topGenes$logFC
y = limma_res$topGenes$adj.P.Val
TCGAVisualize_volcano(
  x,
  y,
  xlab = "logFC",
  title = "Volcano plot of top 1000 genes",)
```

```
## Saving file as: volcano.pdf
```

```
chunk4_time = (proc.time() - ptm)[3] # stop the timer
```

Step 4 - Classification model training, testing, and evaluation

```r
ptm <- proc.time()  # start the timer

# TODO need to redo this whole step using WGCNA

# use the expression data that has been normalized
# Transpose and make it into a matrix object
d_mat = as.matrix(t(limma_res$voomObj$E))

# and the clinical feature to distinguish cases ("definition")
# Make it a factor
d_resp = as.factor(limma_res$voomObj$targets$definition)

# Divide data into training and testing set
# 75% of samples for training and 25% for testing

# Set (random-number-generator) seed so that results are consistent between runs
set.seed(42)

# create a vector of booleans to subset the cases
train_ids = createDataPartition(d_resp, p=0.75, list=FALSE)

# x is the matrix with normalized expression data
# y is the vector with the response variable (tumor vs normal)
x_train = d_mat[train_ids, ]
x_test  = d_mat[-train_ids, ]

y_train = d_resp[train_ids]
y_test  = d_resp[-train_ids]

# do an elastic net model - a generalized linear model that
#  combines lasso and ridge regression, it selects the genes or groups of genes
#  that best predict the condition and uses these to build the model
#  that is then used for classification

# Train model on training dataset using cross-validation
#  alpha can be between 0 (ridge regression) and 1 (lasso)
# the res object here is an object that holds the model coeffiecients and the
#  mean error found during training
res = cv.glmnet(
  x = x_train,
  y = y_train,
  alpha = 0.5,
  family = "binomial")

# Test/Make prediction on test dataset
y_pred = predict(res, newx=x_test, type="class", s="lambda.min")

# confusion matrix shows the TP, TN, FP, and FN
confusion_matrix = table(y_pred, y_test)

# Evaluation statistics
print(confusion_matrix)
```

```
##                       y_test
## y_pred                 Primary solid Tumor Solid Tissue Normal
##   Primary solid Tumor                 103                   1
##   Solid Tissue Normal                   0                   3
```

```
print(paste0("Sensitivity: ",sensitivity(confusion_matrix)))
```

```
## [1] "Sensitivity: 1"
```

```
print(paste0("Specificity: ",specificity(confusion_matrix)))
```

```
## [1] "Specificity: 0.75"
```

```
print(paste0("Precision: ",precision(confusion_matrix)))
```

```
## [1] "Precision: 0.990384615384615"
```

```
# now we can look at the genes that most contribute for the prediction
res_coef = coef(res, s="lambda.min") # the "coef" function returns a sparse matrix

# ignore zero value coefficients
res_coef = res_coef[res_coef[,1] != 0,]

# remove first coefficient as this is the intercept, a variable of the model itself
res_coef = res_coef[-1]

relevant_genes = names(res_coef) # get names of the (non-zero) variables.
length(relevant_genes) # number of selected genes
```

```
## [1] 83
```

```
# get the Ensembl gene names
head(relevant_genes) # few select genes
```

```
## [1] "ENSG00000034971.17" "ENSG00000078804.13" "ENSG00000081181.8"
## [4] "ENSG00000086991.13" "ENSG00000101057.16" "ENSG00000102683.8"
```

```
# get the common gene names
head(limma_res$voomObj$genes)
```

```
##                     source type score phase            gene_id      gene_type
## ENSG00000000003.15 HAVANA gene    NA    NA ENSG00000000003.15 protein_coding
## ENSG00000000005.6  HAVANA gene    NA    NA  ENSG00000000005.6 protein_coding
## ENSG00000000419.13 HAVANA gene    NA    NA ENSG00000000419.13 protein_coding
## ENSG00000000457.14 HAVANA gene    NA    NA ENSG00000000457.14 protein_coding
## ENSG00000000460.17 HAVANA gene    NA    NA ENSG00000000460.17 protein_coding
## ENSG00000000938.13 HAVANA gene    NA    NA ENSG00000000938.13 protein_coding
##                     gene_name level    hgnc_id          havana_gene
## ENSG00000000003.15    TSPAN6     2 HGNC:11858 OTTHUMG00000022002.2
```

```
## ENSG00000000005.6       TNMD     2 HGNC:17757 OTTHUMG00000022001.2
## ENSG00000000419.13      DPM1     2  HGNC:3005 OTTHUMG00000032742.2
## ENSG00000000457.14     SCYL3     2 HGNC:19285 OTTHUMG00000035941.6
## ENSG00000000460.17  C1orf112     2 HGNC:25565 OTTHUMG00000035821.9
## ENSG00000000938.13       FGR     2  HGNC:3697 OTTHUMG00000003516.3
```

```
relevant_gene_names = limma_res$voomObj$genes[relevant_genes,"gene_name"]
head(relevant_gene_names) # few select genes (with readable names now)
```

```
## [1] "MYOC"     "TP53INP2" "ARG2"     "NOX4"     "MYBL2"    "SGCG"
```

```
# did elastic net find the same genes originally found by the limma pipeline?
#  "Of note, we do not expect a high overlap between genes selected by limma and Elastic net.
#   The reason for this is the fact Elastic Net criteria bias the selection of genes,
#   which are not highly correlated against each other, while not such bias is
#   present in limma."
print(intersect(limma_res$topGenes$gene_name, relevant_gene_names))
```

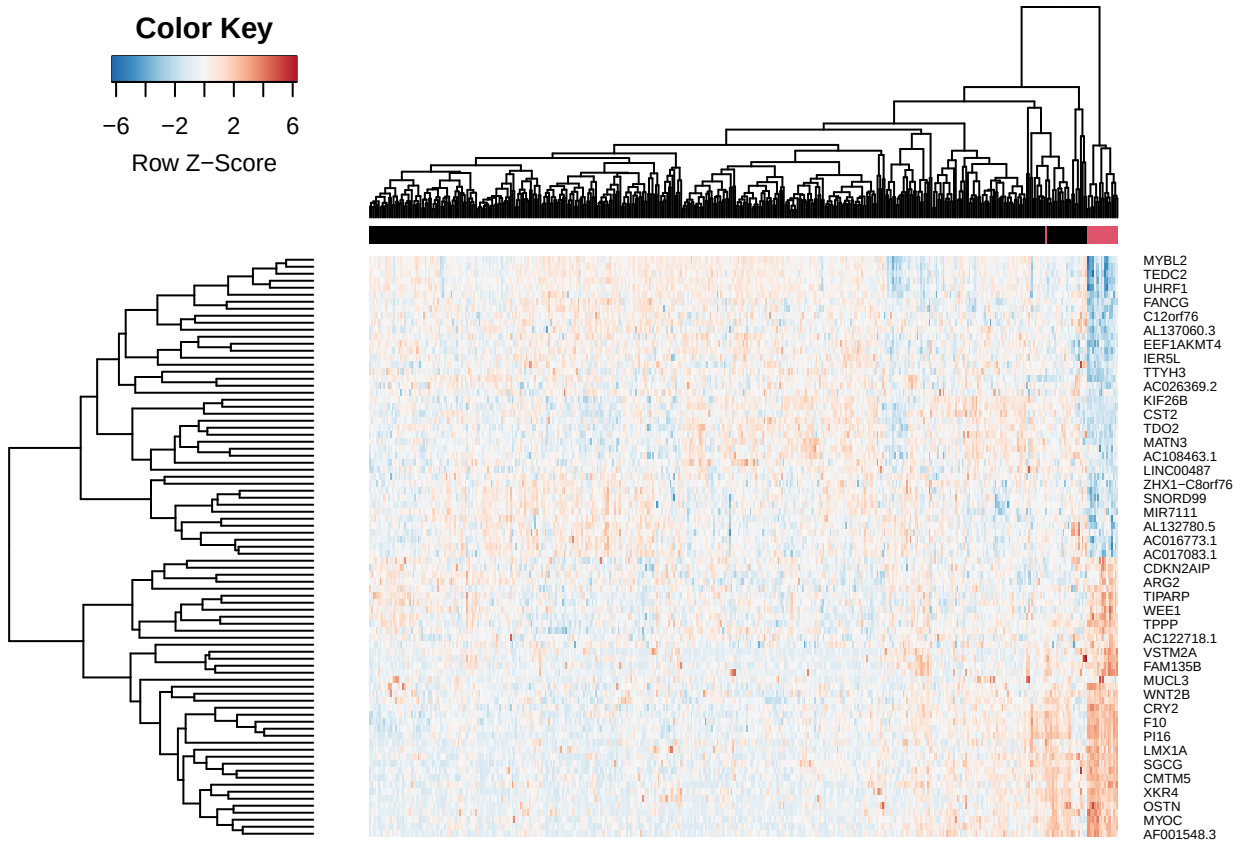
```
##  [1] "PI16"        "CLEC3B"      "CFD"         "F10"         "MYOC"
##  [6] "TPPP"        "AL354861.3"  "XKR4"        "CRY2"        "AF001548.3"
## [11] "PMP2"        "CMTM5"       "NPAS4"       "OSTN"        "LRRC3B"
## [16] "SGCG"        "LMX1A"       "FBXL21P"     "PER2"        "AL161457.1"
## [21] "LRRTM1"      "FAM135B"     "VSTM2A"      "AL669970.3"  "C5orf66-AS1"
## [26] "TP53INP2"    "WNT2B"       "KLF4"        "TEDC2"       "TROAP"
## [31] "THSD4"       "TIPARP"      "UHRF1"       "ATP5MC1P4"   "IQGAP3"
## [36] "ZNF695"      "RNASEH2A"    "LINC01346"   "FANCG"       "AL137060.3"
## [41] "C12orf76"    "WEE1"        "ESM1"
```

```
chunk5_time = (proc.time() - ptm)[3] # stop the timer
```

Step 5 - Hierarchical clustering

```
# we are only considering the elastic net results to cluster genes together
# genes in green are original limma results
# genes in red are normal tissue from the elastic net results
# genes in black are tumor tissue from the elastic net results

ptm <- proc.time()  # start the timer

 # define the color palette for the plot
hmcol = colorRampPalette(rev(brewer.pal(9, "RdBu")))(256)

# perform complete linkage clustering
clust = function(x) hclust(x, method="complete")
# use the inverse of correlation as distance.
dist = function(x) as.dist((1-cor(t(x)))/2)

# Show green color for genes that also show up in DE analysis
colorLimmaGenes = ifelse(
  # Given a vector of boolean values
```

```
  (relevant_genes %in% limma_res$topGenes$ensembl_gene_id),
  "green", # if true, return green for that value
  "white" # if false, return white for that value
)

# generate the heatmap
gene_heatmap = heatmap.2(
  t(d_mat[,relevant_genes]),
  scale="row",          # scale the values for each gene (row)
  density.info="none",  # turns off density plot inside color legend
  trace="none",         # turns off trace lines inside the heat map
  col=hmcol,            # define the color map
  labRow=relevant_gene_names, # use gene names instead of ensembl annotation
  RowSideColors=colorLimmaGenes,
  labCol=FALSE,         # Not showing column labels
  ColSideColors=as.character(as.numeric(d_resp)), # Show colors for each response class
  dendrogram="both",    # Show dendrograms for both axis
  hclust = clust,       # Define hierarchical clustering method
  distfun = dist,       # Using correlation coefficient for distance function
  cexRow=.6,            # Resize row labels
  margins=c(1,5)        # Define margin spaces
)
```




```
chunk6_time = (proc.time() - ptm)[3] # stop the timer
```

Print out the timer values

```
cat(paste0(
  "Chunk 1 finished in ", format(round(chunk1_time, 1), nsmall = 0), "s\n",
  "Chunk 2 finished in ", format(round(chunk2_time, 1), nsmall = 0), "s\n",
  "Chunk 3 finished in ", format(round(chunk3_time, 1), nsmall = 0), "s\n",
  "Chunk 4 finished in ", format(round(chunk4_time, 1), nsmall = 0), "s\n",
  "Chunk 5 finished in ", format(round(chunk5_time, 1), nsmall = 0), "s\n",
  "Chunk 6 finished in ", format(round(chunk6_time, 1), nsmall = 0), "s\n",
  "Total run time: ", format(round(chunk1_time+chunk2_time+
    chunk3_time+chunk4_time+chunk5_time+chunk6_time, 1), nsmall = 0), "s"))
```

```
## Chunk 1 finished in 3.2s
## Chunk 2 finished in 69.5s
## Chunk 3 finished in 12.9s
## Chunk 4 finished in 9.9s
## Chunk 5 finished in 5.1s
## Chunk 6 finished in 0.3s
## Total run time: 100.9s
```