GSE22544 Series: Bootstrapping Technique

*This R script shows the process to obtain significant genes, binarize them, and doa 1000x bootstrap followed by using a distance matrix/heat map and confusion matrix to obtain the F-score, recall, precision.*

### Setting the Working Directory

setwd("D:/Code/RE/My R scripts")

# (1) Preparing the GSE22544 data frame for use in script

Read data frame that contains exprs data:

new\_df\_GSE22544 <- read.csv("new\_df\_GSE22544.csv", header = T, sep = ",", stringsAsFactors = F, row.names = 1) # read exprs dataframe

Rearranging the columns to show the “normal” class as the first 4 columns, and the “IDC” class as the last 16 columns:

new\_df\_GSE22544 <- cbind(new\_df\_GSE22544[, c(4,8,9,13)], new\_df\_GSE22544[,-c(4,8,9,13)]) # rearranges the normal classes to the first 4 columns

Removing any rows that contain NAs:

new\_df\_GSE22544 <- na.omit(new\_df\_GSE22544) # removes NAs which causes probs with for loop

# (2) Method 1: Using for loops to produce binary matrix (but leads to error that says “data are essentially constant”)

Initialising the vector, list, and progress bar library:

significantgenes <- c() # to initialise the vector  
boot\_list <- list()  
library(progress)  
pb <- progress\_bar$new(total = 1000)

Nested for loop to obtain binary vectors of significant genes by resampling 1000x:

for(j in 1:1000){ # 1000 is the number of bootstraps ## cannot be run if done on bigger datasets  
 significantgenes <- c()  
 boot\_normal <- sample(new\_df\_GSE22544[, 1:4], size=4, replace=TRUE)  
 boot\_IDC <- sample(new\_df\_GSE22544[,5:20], size=4, replace=TRUE)  
   
 for(i in 1:2000){ # 9994 is the number of genes # if set to 9995, "data are essentially constant" error  
 ttest\_1 <- try(t.test(boot\_normal[i,], boot\_IDC[i,]))  
 if (ttest\_1$p.value <= 0.05) # if p value is <= 0.05(less than or equals),  
 {  
 significantgenes <- append(significantgenes, 1) # assign the genes (rows) as 1 (significant)  
 }  
 else # if p value is > 0.05,  
 {  
 significantgenes <- append(significantgenes, 0) # assign the genes (rows) as 0 (not significant)  
 }  
 }  
 boot\_list <- append(boot\_list, list(significantgenes))  
 pb$tick()  
 Sys.sleep(1 / 1000)  
 }

# (2) Method 2: Using “genefilter” package’s “rowttests” function (runs faster than nested loop & w/o error)

“genefilter” package is installed and loaded as a library:

#BiocManager::install("genefilter")  
library(genefilter)

Creating a class factor consisting of 2 levels: “normal” and “IDC”:

class\_factor\_2 <- as.factor(c(rep("normal", 4), rep("IDC", 4))) # creating a class factor (normal & IDC)

Initialising the vector, list, and progress bar library:

significantgenes\_2 <- c()  
boot\_list\_2 <- list()  
library(progress)  
pb <- progress\_bar$new(total = 1000)

Utitlising the rowttests() function in a for loop to generate binary matrix:

for (i in 1:1000){  
 m\_normal <- as.matrix(sample(new\_df\_GSE22544[,1:4], size = 4, replace = T)) # samples 4 normal patients into a matrix  
 m\_IDC <- as.matrix(sample(new\_df\_GSE22544[,5:20], size = 4, replace = T)) # samples 4 normal patients into a matrix  
 m\_bind <- cbind(m\_normal, m\_IDC) # has 8 columns, first 4 columns are sampled normal, last 4 columns are sample IDC  
   
 ttest\_2 <- rowttests(m\_bind, class\_factor\_2) # rowttests tests all 9994 rows(genes) w/o error  
 significantgenes\_2 <- as.numeric(ttest\_2$p.value < 0.05) # <0.05 forms a boolean output, and it is changed to numeric by as.numeric.  
 boot\_list\_2 <- append(boot\_list\_2, list(significantgenes\_2)) # append each sampling as a list to a list  
   
 pb$tick() # for progress bar  
 Sys.sleep(1 / 1000) # for progress bar  
}

# (3) Converting list into matrix

The unlist() function, followed by the matrix function converts a list into a matrix:

boot\_mat\_2 <- matrix(unlist(boot\_list\_2), ncol = 1000, byrow = FALSE) # converting list into matrix  
# dim(boot\_mat\_2)

# (4) Computing frequency of significance out of 1000 for each gene (row)

Assigning the sum for each gene (row) to a vector:

sum\_vect <- rowSums(boot\_mat\_2)  
tail(sort(sum\_vect), 5) # [1] 572 590 590 849 906

“for” loop that will generate binary vector to be used as observations for confusion matrix

library(progress)  
pb <- progress\_bar$new(total = length(sum\_vect)) # length of sum\_vect is 9994  
  
for (i in sum\_vect){   
 sum\_sig <- as.numeric(x>180) # as.numeric function converts boolean output to 0s and 1s  
   
 pb$tick() # for progress bar  
 Sys.sleep(1 / length(sum\_vect)) # for progress bar  
}

# (5) Creating confusion matrix

Conversion into factor because confusionMatrix() function requires factors

first\_sample <- as.factor(boot\_mat\_2[,1]) # actual factor  
sum\_sig <- as.factor(sum\_sig) # observation factor

Installation of “caret” package which contains the confusionMatrix() function:

# install.packages("caret")  
library(caret) # for confusion matrix function

Output of Confusion Matrix and assigning the table result to a new variable:

confusionMatrix(sum\_sig, first\_sample)  
 # Confusion Matrix and Statistics  
   
 # Reference  
 # Prediction 0 1  
 # 0 9025 (TP) 324 (FP)  
 # 1 546 (FN) 99 (TN)  
 #   
 # Accuracy : 0.9129   
 # 95% CI : (0.9072, 0.9184)  
 # No Information Rate : 0.9577   
 # P-Value [Acc > NIR] : 1   
 #   
 # Kappa : 0.1415   
 #   
 # Mcnemar's Test P-Value : 6.752e-14   
 #   
 # Sensitivity : 0.9430   
 # Specificity : 0.2340   
 # Pos Pred Value : 0.9653   
 # Neg Pred Value : 0.1535   
 # Prevalence : 0.9577   
 # Detection Rate : 0.9030   
 # Detection Prevalence : 0.9355   
 # Balanced Accuracy : 0.5885   
 #   
 # 'Positive' Class : 0   
  
str(conf\_mat) # $table contains confusion matrix  
(confusionMatrix(sum\_sig, first\_sample))$table  
conf\_mat <- (confusionMatrix(sum\_sig, first\_sample))$table

# (6) Calculating metrics: Precision, Recall, F-Score

Metrics calculated: Precision, Recall, F-Score

# Precision: TP/(TP+FP):  
precision <- conf\_mat[1,1]/sum(conf\_mat[1,1:2])  
precision # [1] 0.9653439  
  
# Recall: TP/(TP + FN):  
recall <- conf\_mat[1,1]/sum(conf\_mat[1:2,1])  
recall # [1] 0.9429527  
  
# F-Score: 2 \* precision \* recall /(precision + recall):  
f\_score <- 2 \* precision \* recall / (precision + recall)  
# [1] 0.9540169  
  
# Metrics  
# Precision: 0.9653439  
# Recall: 0.9429527  
# F-score: 0.9540169

Visualisation of the Confusion Matrix:

cm <-confusionMatrix(sum\_sig, first\_sample)  
  
?confusionMatrix  
draw\_confusion\_matrix <- function(cm) {  
   
 layout(matrix(c(1,1,2)))  
 par(mar=c(2,2,2,2))  
 plot(c(100, 345), c(300, 450), type = "n", xlab="", ylab="", xaxt='n', yaxt='n')  
 title('CONFUSION MATRIX', cex.main=2.5)  
   
 # create the matrix   
 rect(150, 430, 240, 370, col='#3F97D0')  
 text(195, 438, 'Positive', cex=2)  
 rect(250, 430, 340, 370, col='#F7AD50')  
 text(295, 438, 'Negative', cex=2)  
 text(125, 370, 'Predicted', cex=2, srt=90, font=2)  
 text(245, 450, 'Actual', cex=2, font=2)  
 rect(150, 305, 240, 365, col='#F7AD50')  
 rect(250, 305, 340, 365, col='#3F97D0')  
 text(143, 400, 'Positive', cex=2, srt=90)  
 text(143, 335, 'Negative', cex=2, srt=90)  
   
 # add in the cm results   
 res <- as.numeric(cm$table)  
 text(195, 400, res[1], cex=4, font=2, col='black')  
 text(195, 335, res[2], cex=4, font=2, col='black')  
 text(295, 400, res[3], cex=4, font=2, col='black')  
 text(295, 335, res[4], cex=4, font=2, col='black')  
   
 # add in the specifics   
 plot(c(100, 0), c(100, 0), type = "n", xlab="", ylab="", main = "DETAILS", xaxt='n', yaxt='n')  
 text(10, 85, names(cm$byClass[1]), cex=1.2, font=2)  
 text(10, 70, round(as.numeric(cm$byClass[1]), 3), cex=1.2)  
 text(30, 85, names(cm$byClass[2]), cex=1.2, font=2)  
 text(30, 70, round(as.numeric(cm$byClass[2]), 3), cex=1.2)  
 text(50, 85, names(cm$byClass[5]), cex=1.2, font=2)  
 text(50, 70, round(as.numeric(cm$byClass[5]), 3), cex=1.2)  
 text(70, 85, names(cm$byClass[6]), cex=1.2, font=2)  
 text(70, 70, round(as.numeric(cm$byClass[6]), 3), cex=1.2)  
 text(90, 85, names(cm$byClass[7]), cex=1.2, font=2)  
 text(90, 70, round(as.numeric(cm$byClass[7]), 3), cex=1.2)  
   
 # add in the accuracy information   
 text(30, 35, names(cm$overall[1]), cex=1.5, font=2)  
 text(30, 20, round(as.numeric(cm$overall[1]), 3), cex=1.4)  
 text(70, 35, names(cm$overall[2]), cex=1.5, font=2)  
 text(70, 20, round(as.numeric(cm$overall[2]), 3), cex=1.4)  
}   
  
draw\_confusion\_matrix(cm)

# (7) Compute a distance matrix using Jaccard’s coefficient

Create user-defined function that calculates jaccard coefficient (intersection over union):

jaccard\_fun <- function (x,y) {   
 M.11 = sum(x == 1 & y == 1)  
 M.10 = sum(x == 1 & y == 0)  
 M.01 = sum(x == 0 & y == 1)  
 return (M.11 / (M.11 + M.10 + M.01))  
}

Initialising dataframe for imputation:

jaccard\_df <- as.data.frame(matrix(NA,nrow = 1000, ncol = 1000))  
names(jaccard\_df) <- paste0('S', 1:1000)  
rownames(jaccard\_df) <- paste0('S', 1:1000)

For loop that will produce the distance matrix:

library(progress)  
pb <- progress\_bar$new(total = 1000)  
  
for (r in 1:1000) {  
 for (c in 1:1000) {  
 if (c == r) { # if rows iteration is the same as column iteration,  
 jaccard\_df[r,c] = 1 # assign as 1  
 } else if (c > r) { # if not then when columns is more than rows, add the variables of rows and  
 jaccard\_df[r,c] <- jaccard\_fun(boot\_mat\_2[,r], boot\_mat\_2[,c]) # replace with list from above  
 }  
 }  
 pb$tick() # for progress bar  
 Sys.sleep(1 / 1000) # for progres bar  
}

# Writing data into files for further use

write.csv(boot\_mat\_2, file = "D:/Code/RE/My R scripts/boot\_mat\_2.csv", row.names = T)  
write.csv(jaccard\_df, file = "D:/Code/RE/My R scripts/jaccard\_df.csv", row.names = T)