#Libraries

```{r}

library(gplots)#heatmap.2

library(ggplot2)

library(genefilter)

library(edgeR)

library(reshape2)

library(moments) #calculate skewness and kurtosis

library(gridExtra) #plots side by side

library(factoextra) #prcomp visualization

library(VIM) #missing data graphics

library(LMGene) #glog

library(limma) #batch effects

library(pheatmap)

library(ggfortify) #pca autoplot

library(FactoMineR)

library(devtools)

library(ggbiplot)

library(factoextra)

library(mixOmics)

library(vsn)

```

#s in the file are erased manually before reading it into R

#Read file

```{r}

df <- read.table("Kallisto\_ELS\_JQ\_EE2.txt", header = T, as.is = T)

grp.lbl <- as.factor(df[1,-1])

grp.nms <- levels(grp.lbl)

df <- data.matrix(df[-1, ])

rownames(df) <- as.character(df[,1])

df <- df[ ,-1]

df2 <- read.table("Kallisto\_Adult\_JQ\_EE2.txt", header = T, as.is = T)

grp.lbl2 <- as.factor(df2[1,-1])

grp.nms2 <- levels(grp.lbl2)

df2 <- data.matrix(df2[-1, ])

rownames(df2) <- as.character(df2[,1])

df2 <- df2[ ,-1]

df2 <- df2[,-16] #one more control

grp.lbl2 <- grp.lbl2[-16] #one more control

#Reorder to match ELS

df2.high <- df2[,1:5]

df2.med <- df2[,6:10]

df2.ctl <- df2[,11:15]

grp.lbl2.high <- grp.lbl2[1:5]

grp.lbl2.med <- grp.lbl2[6:10]

grp.lbl2.ctl <- grp.lbl2[11:15]

df2 <- cbind(df2.ctl, df2.med, df2.high)

grp.lbl2 <- factor(c(grp.lbl2.ctl, grp.lbl2.med, grp.lbl2.high), labels=c("Control","High","Medium"))

```

#If row names of data contain duplicates, their values will be replaced by the sum of all duplicates

```{r}

SumDuplicates <- function(data){

all.nms <- rownames(data);

dup.inx <- duplicated(all.nms);

if(sum(dup.inx) > 0){

uniq.nms <- all.nms[!dup.inx];

uniq.data <- data[!dup.inx,,drop=F];

dup.nms <- all.nms[dup.inx];

uniq.dupnms <- unique(dup.nms);

uniq.duplen <- length(uniq.dupnms);

for(i in 1:uniq.duplen){

nm <- uniq.dupnms[i];

hit.inx.all <- which(all.nms == nm);

hit.inx.uniq <- which(uniq.nms == nm);

uniq.data[hit.inx.uniq, ] <- apply(data[hit.inx.all,,drop=F], 2, sum);

}

return(uniq.data);

}else{

return(data);

}

}

df\_dup\_fixed <- SumDuplicates(df)

df\_dup\_fixed <- t(df\_dup\_fixed)

#To check the duplicates manually

write.table(df\_dup\_fixed, "df\_dup\_fixed.txt", sep = "\t", row.names = TRUE)

#double check duplication

sum(duplicated(colnames(df\_dup\_fixed)))

df\_dup\_fixed2 <- SumDuplicates(df2)

df\_dup\_fixed2 <- t(df\_dup\_fixed2)

sum(duplicated(colnames(df\_dup\_fixed2)))

```

#Missing value check

```{r}

miss\_check <- function(data1){

if(sum(colSums(is.na(data1))) != 0){

warning("Data1 and Data2 have the missing values. Need missing data imputations.")

}else{

print("All good.")

}

}

miss\_check(df\_dup\_fixed)

miss\_check(df\_dup\_fixed2)

#Can show the histograms using VIM package. - need to make user interactive

#try(VIM::aggr(df\_dup\_fixed, col=c('blue','red'), numbers=TRUE, sortVars=FALSE,

#labels=names(df\_dup\_fixed), cex.axis=.5, gap=3, ylab=c("Histogram of missing data","Pattern"),

#xlab = c("variables", "variables")))

```

# Filter

# relevant library: genefilter

# filter out unannotated genes

# ref: <http://web.mit.edu/~r/current/arch/i386_linux26/lib/R/library/genefilter/html/nsFilter.html>

# Info on **eset** for the above functions: <https://www.rdocumentation.org/packages/Biobase/versions/2.32.0/topics/ExpressionSet>

# filter out genes with **low variance and low abundance**

# ref: <https://rdrr.io/github/yunzhang813/rGTExNetwork/man/geneFilter.html> (not used below)

# filter out the top 40%

# the df: sample type as rows; genes as columns

# output is the column index of the genes remained after filtering

It looks reasonable to me to use mean abundance(=count per million, I guess) “per sample” as genefilter does. (I am also not familiar with these!) The reason is that each sample can be considered as an experiment that generates different number of counts overall. So, using mean for each sample may have effect as erasing difference between samples, I guess.

According to a query, authors of genefilter package worked together to make DESeq2 package so DESeq2 automatically do filter genes within the package. However, EdgeR does not have such filtering system within it. These kinds of filtering before DEA helps increasing the stat power but some say it is not mandatory.

-> this is the algorithm below. So, it becomes user interactive, and you can decide what to go for amongst those 5 options. If you want to use 2 or more, you can simply run them twice…. Done!!!

# <https://statquest.org/2017/05/16/statquest-filtering-genes-with-low-read-counts/>

#Low count filter - features with very small counts in very few samples are likely due to sequencing errors or low-level contaminations. You need to first specify a minimum count (default 4). For instance, if you use 20% prevalence filter, meaning for any feature to be retained, at least 20% of its values should contain at least 4 counts. You can also filter low abundance features based on whether their mean or median values are below the minimum count.

#Low variance filter - features that are close to constant throughout the experiment conditions are unlikely to be associated with the conditions under study (useful for comparative analysis). Their variances can be measured using inter-quantile range (IQR), standard deviation or coefficient of variation (CV). The lowest percentage based on the specified cutoff will be excluded.

```{r}

filter\_indx <- function(n, data){

if (n == 1){

#calculate the abs maximum of gene expression level (GE) per each gene and take the top 40% -

prop <- readline(promp = "Enter the proportion [0, 1] to remove: ")

prop <- as.numeric(prop)

maxGE <- apply(data, 2, function (x) max(abs(x)))

propGEmax <- quantile(maxGE, prop, na.rm = T)

indx <- which(maxGE > propGEmax)

}else if (n == 2){

#take the IQR of each gene and take the top genes 40%: low variance -

prop <- readline(promp = "Enter the proportion [0, 1] to remove: ")

prop <- as.numeric(prop)

IQRGE <- apply(data, 2, IQR, na.rm=T)

propGEIQR <- quantile(IQRGE, prop, na.rm = T)

indx <- which(IQRGE> propGEIQR)

}else if (n == 3){

#take the standard deviation of each gene and take the top genes 40%: low variance -

prop <- readline(promp = "Enter the proportion [0, 1] to remove: ")

prop <- as.numeric(prop)

sdGE <- apply(data, 2, sd, na.rm=T)

sdGEIQR <- quantile(sdGE, prop, na.rm = T)

indx <- which(sdGE> sdGEIQR)

}else if (n == 4){

#take the standard deviation of each gene and take the top genes 40%: low variance -

prop <- readline(promp = "Enter the proportion [0, 1] to remove: ")

prop <- as.numeric(prop)

sds <- apply(data, 2, sd, na.rm=T)

mns <- apply(data, 2, mean, na.rm=T)

coefs <- abs(sds/mns)

coefsGEIQR <- quantile(coefs, prop, na.rm = T)

indx <- which(coefs> coefsGEIQR)

}else if (n == 5){

#low count of sum - DESeq2

count\_threshold <- readline(promp = "Enter the count threshold: ")

count\_threshold <- as.numeric(count\_threshold)

sums <- apply(data, 2, sum, na.rm = T)

indx <- sums >= count\_threshold

}else if (n == 6){

#low count filter with mean abundance - DESeq2

count\_threshold <- readline(promp = "Enter the count threshold: ")

count\_threshold <- as.numeric(count\_threshold)

meanGE <- apply(data, 2, mean, na.rm = T)

indx <- meanGE >= count\_threshold

}else if (n == 7){

#low count filter with median abundance - DESeq2

count\_threshold <- readline(promp = "Enter the count threshold: ")

count\_threshold <- as.numeric(count\_threshold)

medGE <- apply(data, 2, median, na.rm = T)

indx <- medGE >= count\_threshold

}else {

#low count filter with prevalence - not recommended - EdgeR method

count\_threshold <- readline(promp = "Enter the count threshold: ")

count\_threshold <- as.numeric(count\_threshold)

prev <- readline(promp = "Enter the prevalence [0, 1]: ")

prev <- as.numeric(prev)

minLen <- prev \* nrow(data)

indx <- apply(data, 2, function(x) {sum(x >= count\_threshold) >= minLen})

}

return(indx)

}

# low variance: n = 2, option = 0.2

n <- readline(prompt="Enter an integer (1. absolute maximum of gene expression 2. Low variance based on IQR 3. Low variance based on standard deviation 4. Low variance based on coefficient of variation 5. low count of sum 6. low count with mean abundance 7. low count with median abundance 8. low count with prevalence in sample %): ")

n <- as.numeric(n)

keep\_inx <- filter\_indx(n, df\_dup\_fixed)

df\_dup\_fixed\_filt <- df\_dup\_fixed[,keep\_inx]

keep\_inx2 <- filter\_indx(n, df\_dup\_fixed2)

df\_dup\_fixed\_filt2 <- df\_dup\_fixed2[,keep\_inx2]

# low count: n = 6, option = 10

n <- readline(prompt="Enter an integer (1. absolute maximum of gene expression 2. Low variance based on IQR 3. Low variance based on standard deviation 4. Low variance based on coefficient of variation 5. low count of sum 6. low count with mean abundance 7. low count with median abundance 8. low count with prevalence in sample %): ")

n <- as.numeric(n)

keep\_inx <- filter\_indx(n, df\_dup\_fixed\_filt)

df\_dup\_fixed\_filt <- df\_dup\_fixed\_filt[,keep\_inx]

keep\_inx2 <- filter\_indx(n, df\_dup\_fixed\_filt2)

df\_dup\_fixed\_filt2 <- df\_dup\_fixed\_filt2[,keep\_inx2]

#dim(df\_dup\_fixed[,filter\_indx(n, df\_dup\_fixed)]) #check the answer

```

#Data dimension check

```{r}

data\_statistics <- data.frame(matrix(ncol = 2, nrow = 3))

x <- c("#rows/sample", "#columns/feature")

colnames(data\_statistics) <- x

y <- c("original", "duplicate\_removed", "filtered")

rownames(data\_statistics) <- y

data\_statistics[1,] <- c(nrow(t(df)), ncol(t(df)))

data\_statistics[2,] <- c(nrow(df\_dup\_fixed), ncol(df\_dup\_fixed))

data\_statistics[3,] <- c(nrow(df\_dup\_fixed\_filt), ncol(df\_dup\_fixed\_filt))

data\_statistics

data\_statistics.adult <- data.frame(matrix(ncol = 2, nrow = 3))

x <- c("#rows/sample", "#columns/feature")

colnames(data\_statistics.adult) <- x

y <- c("original", "duplicate\_removed", "filtered")

rownames(data\_statistics.adult) <- y

data\_statistics.adult[1,] <- c(nrow(t(df2)), ncol(t(df2)))

data\_statistics.adult[2,] <- c(nrow(df\_dup\_fixed2), ncol(df\_dup\_fixed2))

data\_statistics.adult[3,] <- c(nrow(df\_dup\_fixed\_filt2), ncol(df\_dup\_fixed\_filt2))

data\_statistics.adult

```

#### Normalization ####

<http://ncss-tech.github.io/stats_for_soil_survey/chapters/4_exploratory_analysis/4_exploratory_analysis.html>

<https://towardsdatascience.com/visualizing-your-exploratory-data-analysis-d2d6c2e3b30e>

#Normalization

auto-scaling (HW2) - checked

generalized log transformation (o2pls) - checked

min-max scaling (HW2) - checked

two-parameter box-cox transformation (o2pls) - checked

upper quantile normalization (ecotoxx) - checked

trimmed mean of M-values normalization (ecotoxx) - checked

log2 counts per million (ecotoxx) - checked

#normalization method TMM is used in this paper: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0139324>

```{r}

n <- readline(prompt="Enter an integer (1. auto scaling 2. generalized log transformation

3. min-max scaling 4. two-parameter box cox transformation

5. quantile normalization 6. variancne stabilizing normalization

7. log2 transformation 8. log2 counts per million

9. trimmed mean of M-values normalization

10. relative log expression normalisation

11. upper quantile normalization

12. variance stabilizing normalization and quantile normalization): ") #any method works fine

n <- as.numeric(n)

normalized <- function(n, data){

if (n == 1){

#auto scaling

normed <- scale(data)

} else if (n == 2){

#generalized log transformation

lmbd <- readline(promp = "Enter the lambda: ")

lmbd <- as.numeric(lmbd)

normed <- glog(data, lmbd)

} else if (n == 3){

#min-max scaling

min\_max <- function(x){

return ((x - min(x)) / (max(x) - min(x)))

}

normed <- apply(df\_dup\_fixed\_filt, 2, min\_max)

} else if (n == 4){

#two-parameter box-cox transformation

lambd1 <- readline(prompt="Enter a lambda1: ")

lambd1 <- base::eval(parse(text = lambd1))

lambd2 <- readline(prompt="Enter a lambda2: ")

lambd2 <- base::eval(parse(text = lambd2))

if(all(data > -lambd2)){

if(lambd1 == 0){

normed <- log(data+lambd2)

}else{

normed <- ((data+lambd2)^(lambd1) - 1)/lambd1

}

}else{

stop("some of the data <= -lambda2")

}

} else if (n == 5){

#quantile normalization

library(preprocessCore)

normed <- normalize.quantiles(t(data), copy=TRUE)

normed <- t(normed)

} else if (n == 6){

#variancne stabilizing normalization

library(limma)

library(vsn)

normed <- normalizeVSN(t(data), minDataPointsPerStratum = 3) #change minDataPointsPerStratum

normed <- t(normed)

} else if (n == 7){

#log2 transformation

min.val <- min(data[data>0], na.rm=T)/10

data[data<=0] <- min.val

normed <- log2(data)

} else if (n == 8){

#log2 counts per million

library(edgeR)

nf <- edgeR::calcNormFactors(t(data))

y <- voom(t(data),plot=F,lib.size=colSums(t(data))\*nf)

normed <- y$E

normed <- t(normed)

} else if (n == 9){

#trimmed mean of M-values normalization

library(edgeR)

nf <- edgeR::calcNormFactors(t(data),method="TMM")

y <- voom(t(data),plot=F,lib.size=colSums(t(data))\*nf)

normed <- y$E

normed <- t(normed)

#or,

# suppressMessages(require("edgeR"));

# data[data<0] <- 0

# otuTMM <- edgeRnorm(t(data),method="TMM");

# data <- as.data.frame(t(otuTMM$counts));

} else if (n == 10){

#Relative log expression normalisation

library(edgeR)

min.val <- min(data[data>0], na.rm=T)/10

data[data<=0] <- min.val

nf <- edgeR::calcNormFactors(t(data),method="RLE")

y <- voom(t(data),plot=F,lib.size=colSums(t(data))\*nf)

normed <- y$E

normed <- t(normed)

#or,

# suppressMessages(require("edgeR"));

# data[data<0] <- 0

# otuRLE <- edgeRnorm(t(data),method="RLE");

# data <- as.data.frame(t(otuRLE$counts))

} else if (n == 11){

#upper quantile normalization

library(edgeR)

nf <- edgeR::calcNormFactors(t(data),method="upperquartile")

y <- voom(t(data),plot=F,lib.size=colSums(t(data))\*nf)

normed <- y$E

normed <- t(normed)

#or,

# suppressMessages(require("edgeR"));

# data[data<0] <- 0

# otuUQ <- edgeRnorm(t(data),method="upperquartile");

# data <- as.data.frame(t(otuUQ$counts));

} else {

#variancne stabilizing normalization and quantile normalization

library(limma)

data <- normalizeVSN(t(data), minDataPointsPerStratum = 3) #change minDataPointsPerStratum

library(preprocessCore)

normed <- normalize.quantiles(data, copy=TRUE)

normed <- t(normed)

}

return(normed)

}

df\_dup\_final <- normalized(n, df\_dup\_fixed\_filt)

df\_dup\_final2 <- normalized(n, df\_dup\_fixed\_filt2)

```

#Sanity check

box plot (ecotoxx) - done

density plot (ecotoxx) - done

histogram - done

heatmap (ecotoxx) - done

mean-standard deviation plot (ecotoxx) - done

some tables of statistical values (dimensions, mean, median, mode, summary, str, IQR, etc) - done

scatter plot - done

standard deviation plot (o2pls) - done

‘mean - median’ plot (o2pls) - done

kurtosis plot (o2pls) - done

skewness plot (o2pls) - done

quantile-quantile plot - done

conditioning scatter plot - done

correlation analysis - done

```{r}

n <- readline(prompt="Enter an integer (1. boxplot 2. density plot

3. histogram 4. heatmap

5. mean-standard deviation plot 6. tables

7. scatter plot 8. Normalization check plots

9. Conditioning plot

10. Correlation plot): ")

n <- as.numeric(n)

sanity <- function (n, data){

if (n == 1){

boxplot(t(data), xlab = "counts", ylab = "sample", las = 2, cex.axis = 0.8, horizontal = T, main = "boxplot")

} else if (n == 2){

par(mfrow = c(2, 2))

library(lattice) #densityplot

for (i in 1:nrow(data)){

densityplot(data[i,], main = paste("density ", i, sep = ""))

}

for (i in 1:nrow(data)){

den <- density(data[i,], adjust = 0.4) #bumpy - change adjust

plot(den, main = paste("density ", i, sep = ""))

polygon(den, col = "red", border = "blue")

density(data[i,])

}

} else if (n == 3){

for (i in 1:nrow(data)){

hist(data[i,], main = paste("histogram ", i, sep = ""))

}

} else if (n == 4){

pheatmap(data)

} else if (n == 5){

library(vsn)

meanSdPlot(as.matrix(t(data)), ranks=FALSE) #sample should be on columns

# Diagnostic plot summarizing the standard deviation versus mean measures of reads in the samples for each gene. It checks whether there is a dependence between counts and variance.

} else if (n == 6){

#some tables of statistical values (dimensions, mean, median, mode, summary, str, IQR, etc)

getmode <- function(v) {

uniqv <- unique(v)

uniqv[which.max(tabulate(match(v, uniqv)))]

}

data\_statistics2 <- data.frame(matrix(ncol = 6, nrow = nrow(data)))

x <- c("Min", "Median", "Mean", "Mode", "Max", "IQR")

colnames(data\_statistics2) <- x

y <- rownames(data)

rownames(data\_statistics2) <- y

for (i in 1:nrow(data)){

vecs <- c(min(data[i,]), median(data[i,]), mean(data[i,]), getmode(data[i,]),

max(data[i,]), IQR(data[i,])

)

data\_statistics2[i,] <- vecs

}

data\_statistics2

} else if (n == 7){

par(mfrow = c(2, 2))

for (i in 1:nrow(data)){

plot(data[i,], main = paste("scatter ", i, sep = ""), xlab = "gene", ylab = "count")

}

pairs(df\_dup\_final[,1:10], panel = panel.smooth) #change genes if you want

} else if (n == 8){

for (i in 1:nrow(data)){

par(mfrow = c(1,1))

qqnorm(data[i,], main = rownames(data)[i]); qqline(data[i,])

}

par(mfrow = c(2,2))

plot(apply(data, 2, mean) - apply(data, 2, median), ylab = "Mean - Median", xlab = "gene")

plot(apply(data, 2, sd), ylab = "Stdv", xlab = "gene")

plot(apply(data, 2, skewness), ylab = "Skewness", xlab = "gene")

plot(apply(data, 2, kurtosis), ylab = "Kurtosis", xlab = "gene")

} else if (n == 9){

coplot(data[1,] ~ data[2,] | data[3,], rows = 1, overlap = 0, number = 5) #can change the value

} else {

library(corrplot) #correlation plot

corrplot(cor(data[,1:10], data[,100:120]), method = "circle")#can change the value

corrplot.mixed(cor(data[,220:240]))#can change the value

corrplot(cor(data[,1000:1020]), order = "hclust")#can change the value

corrplot(cor(data[,2000:2020]), order = "hclust", addrect = 2)#can change the value

col3 <- colorRampPalette(c("red", "white", "blue"))

corrplot(cor(data[,4000:4020]), order = "hclust", addrect = 2, col = col3(20))#can change the value

res1 <- cor.mtest(data[,10000:10020], conf.level = .95)#can change the value

corrplot(cor(data[,10000:10020]), p.mat = res1$p, sig.level = .2)#can change the value

corrplot(cor(data[,10000:10020]), p.mat = res1$p, insig = "p-value")#can change the value

corrplot(cor(data[,10000:10020]), p.mat = res1$p, order = "hclust", insig = "pch", addrect = 2) #can change the value

}

}

sanity(n, df\_dup\_final)

sanity(n, df\_dup\_final2)

#Boxplot

par(mfrow = c(1,2))

boxplot(t(df\_dup\_final), xlab = "counts", ylab = "sample", las = 1, cex.axis = 0.5, horizontal = T, main = "ELS boxplot")

boxplot(t(df\_dup\_final2), xlab = "counts", ylab = "sample", las = 1, cex.axis = 0.5, horizontal = T, main = "Adult boxplot")

#Density merged (5 samples for each group -> 1 big vector)

par(mfrow = c(2, 3))

library(lattice) #densityplot

els.con <- c(t(df\_dup\_final[1:5,]))

els.med <- c(t(df\_dup\_final[6:10,]))

els.h <- c(t(df\_dup\_final[11:15,]))

adl.con <- c(t(df\_dup\_final2[1:5,]))

adl.med <- c(t(df\_dup\_final2[6:10,]))

adl.h <- c(t(df\_dup\_final2[11:15,]))

den <- density(els.con, adjust = 0.4) #bumpy - change adjust

plot(den, main = "ELS con")

polygon(den, col = "red", border = "blue")

den <- density(els.med, adjust = 0.4) #bumpy - change adjust

plot(den, main = "ELS med")

polygon(den, col = "red", border = "blue")

den <- density(els.h, adjust = 0.4) #bumpy - change adjust

plot(den, main = "ELS high")

polygon(den, col = "red", border = "blue")

den <- density(adl.con, adjust = 0.4) #bumpy - change adjust

plot(den, main = "ADL con")

polygon(den, col = "red", border = "blue")

den <- density(adl.med, adjust = 0.4) #bumpy - change adjust

plot(den, main = "ADL med")

polygon(den, col = "red", border = "blue")

den <- density(adl.h, adjust = 0.4) #bumpy - change adjust

plot(den, main = "ADL high")

polygon(den, col = "red", border = "blue")

#heatmap

pheatmap(df\_dup\_final)

pheatmap(df\_dup\_final2)

#ms plot

msd <- meanSdPlot(as.matrix(t(df\_dup\_final)), ranks=FALSE)

msd2 <- meanSdPlot(as.matrix(t(df\_dup\_final2)), ranks=FALSE)

msd$gg + ggtitle("ELS")

msd2$gg + ggtitle("Adult")

#table

#some tables of statistical values (dimensions, mean, median, mode, summary, str, IQR, etc)

getmode <- function(v) {

uniqv <- unique(v)

uniqv[which.max(tabulate(match(v, uniqv)))]

}

data\_statistics.els <- data.frame(matrix(ncol = 6, nrow = nrow(df\_dup\_final)))

x <- c("Min", "Median", "Mean", "Mode", "Max", "IQR")

colnames(data\_statistics.els) <- x

y <- rownames(df\_dup\_final)

rownames(data\_statistics.els) <- y

for (i in 1:nrow(df\_dup\_final)){

vecs <- c(min(df\_dup\_final[i,]), median(df\_dup\_final[i,]), mean(df\_dup\_final[i,]), getmode(df\_dup\_final[i,]),

max(df\_dup\_final[i,]), IQR(df\_dup\_final[i,])

)

data\_statistics.els[i,] <- vecs

}

data\_statistics.els

#some tables of statistical values (dimensions, mean, median, mode, summary, str, IQR, etc)

getmode <- function(v) {

uniqv <- unique(v)

uniqv[which.max(tabulate(match(v, uniqv)))]

}

data\_statistics.adult <- data.frame(matrix(ncol = 6, nrow = nrow(df\_dup\_final2)))

x <- c("Min", "Median", "Mean", "Mode", "Max", "IQR")

colnames(data\_statistics.adult) <- x

y <- rownames(df\_dup\_final2)

rownames(data\_statistics.adult) <- y

for (i in 1:nrow(df\_dup\_final2)){

vecs <- c(min(df\_dup\_final2[i,]), median(df\_dup\_final2[i,]), mean(df\_dup\_final2[i,]), getmode(df\_dup\_final2[i,]),

max(df\_dup\_final2[i,]), IQR(df\_dup\_final2[i,])

)

data\_statistics.adult[i,] <- vecs

}

data\_statistics.adult

#scatter

par(mfrow = c(2, 2))

for (i in 1:nrow(df\_dup\_final)){

plot(df\_dup\_final[i,], main = paste("scatter ", i, sep = ""), xlab = "gene", ylab = "count")

}

#pairs(df\_dup\_final[,1:10], panel = panel.smooth) #change genes if you want

#normalization check

for (i in 1:nrow(df\_dup\_final)){

par(mfrow = c(1,1))

qqnorm(df\_dup\_final[i,], main = rownames(df\_dup\_final)[i]); qqline(df\_dup\_final[i,])

}

par(mfrow = c(2,2))

plot(apply(df\_dup\_final, 2, mean) - apply(df\_dup\_final, 2, median), ylab = "Mean - Median", xlab = "gene")

plot(apply(df\_dup\_final, 2, sd), ylab = "Stdv", xlab = "gene")

plot(apply(df\_dup\_final, 2, skewness), ylab = "Skewness", xlab = "gene")

plot(apply(df\_dup\_final, 2, kurtosis), ylab = "Kurtosis", xlab = "gene")

#Conditioning plot

coplot(df\_dup\_final[1,] ~ df\_dup\_final[2,] | df\_dup\_final[3,], rows = 1, overlap = 0, number = 5) #can change the value

```

#### differential expression analysis ####

<http://chagall.med.cornell.edu/RNASEQcourse/Intro2RNAseq.pdf>

# relevant library: edgeR

# should choose statistical method e.g. EdgeR, DESeq2, Limma voom, …

# filter out with significant thresholds of adjusted p-value < 0.05 and abs(logFC) > 1

#Differential expression analysis (ecotoxx)

DESeq2 (ecotoxx) -

edgeR (ecotoxx)

limmaVoom (ecotoxx)

Cuffdiff (ecotoxx)

MA plot (ecotoxx)

volcano plot (ecotoxx)

table including adjusted p-value, log fold change and gene ID (ecotoxx)

up/down regulated (ecotoxx)

DEGs will be determined with adjusted p-value (or false discovery rate, FDR) of 0.05 and logarithmic fold change (log2FC) of 1 using EdgeR (Robinson et al., 2010), which is recommended for a study with less than 12 replicates (Schurch et al., 2016) (ecotoxx)

# Declare a function for differential expression analysis

```{r}

DEAnal<-function (data, n, m, p.lvl, fc.lvl){

set.seed(1)

myargs <- list()

if (n == 1) {

inx = 0

for (m in 1:(length(cls.nms) - 1)) {

for (n in (m + 1):length(cls.nms)) {

inx <- inx + 1

myargs[[inx]] <- paste(cls.nms[m], "-", cls.nms[n], sep = "")

}

}

filename <<- "SigGene\_pairwise"

} else if (n == 2) {

for (i in 2:length(cls.nms)) {

myargs[[i - 1]] <- paste(cls.nms[i], "-", cls.nms[i-1], sep = "")

}

filename <<- "SigGene\_time\_series"

} else if (n == 3) {

cntrl <- "Control" #readline(prompt=paste0("What is the name of control group?(", cls.nms, ") : "))

ref <- readline(prompt=paste0("Which group would you want to compare with the control? (", cls.nms[cls.nms != cntrl], ") : "))

myargs[[1]] <- paste0(ref, "-", cntrl)

filename <<- paste0("SigGene\_", ref, "\_vs\_", cntrl)

} else {

cntrl <- "Control" #readline(prompt=paste0("What is the name of control group?(", cls.nms, ") : "))

ref <- cls.nms[cls.nms != cntrl]

myargs <- as.list(paste0(ref, "-", cntrl))

filename <<- paste0("SigGene\_reference\_", cntrl)

}

library(limma)

design <- model.matrix(~-1 + cls.lbl)

colnames(design) <- cls.nms

myargs[["levels"]] <- design

contrast.matrix <- do.call(makeContrasts, myargs)

block <- NULL

#block <- rep(1:3, each=5) #Block is necessary for a specific comparison set. Please refer to p.50 of the Limma User Guide ver. January 2019.

if (m == 1) {

# Differential analysis with the Limma voom package

if (is.null(block)) {

fit = lmFit(t(data), design)

} else {

corfit <- duplicateCorrelation(t(data), design, block = block)

fit <- lmFit(t(data), design, block = block, correlation = corfit$consensus)

}

if (!is.fullrank(design)) {

current.msg <<- paste("This metadata combination is not full rank! Please use other combination.")

return(0)

}

if (all(fit$df.residual == 0)) {

current.msg <<- paste("There is not enough replicates in each group (no residual degrees of freedom)!")

return(0);

}

fit2 <- contrasts.fit(fit, contrast.matrix)

fit2 <- eBayes(fit2)

topFeatures <- topTable(fit2, number = Inf, adjust.method = "fdr")

} else if (m == 2){

# Differential analysis with the DESeq2 package

library(DESeq2);

if (is.null(dataSet$sec.cls)){

colData <- data.frame(dataSet$fst.cls)

colnames(colData) <- "condition"

dds <- DESeqDataSetFromMatrix(countData=round(dataSet$data.anot), colData = colData, design = ~condition);

} else {

colData <- data.frame(dataSet$fst.cls, dataSet$sec.cls, dataSet$cls);

colnames(colData) <- c("condition", "type", "condition\_type");

dds <- DESeqDataSetFromMatrix(countData=round(dataSet$data.anot), colData = colData, design = ~condition\_type);

}

dds <- DESeq(dds, betaPrior=TRUE)

vec <- as.numeric(c(0, contrast.matrix[,1]))

#vec = vec[1:length(resultsNames(dds))]

res <- results(dds, contrast = vec, independentFiltering = FALSE, cooksCutoff = Inf)

topFeatures <- data.frame(res@listData);

rownames(topFeatures) <- rownames(res);

nms <- colnames(topFeatures);

nms[which(nms == "padj")] <- "adj.P.Val"

nms[which(nms == "pvalue")] <- "P.Value"

nms[which(nms == "log2FoldChange")] <- "logFC"

colnames(topFeatures) <- nms

topFeatures <- topFeatures[c(2,1,3,4,5,6)]

# order the result based on raw p

ord.inx <- order(topFeatures$P.Value)

topFeatures <- topFeatures[ord.inx, ]

} else {

# Differential analysis with the EdgeR package

library(edgeR)

y <- DGEList(counts=t(edger\_df), group=cls.lbl)

keep <- rowSums(cpm(y)>1) >= 3 #It means I am happy to see genes that are differentially expressed in >= 3 samples in a group

y <- y[keep, ]

y <- calcNormFactors(y)

#plotMDS(y)

y <- estimateGLMCommonDisp(y, design, verbose = FALSE)

y <- estimateGLMTrendedDisp(y, design)

y <- estimateGLMTagwiseDisp(y, design)

#plotBCV(y)

fit <- glmFit(y, design)

lrt <- glmLRT(fit, contrast = contrast.matrix)

topFeatures <- topTags(lrt, n = Inf)$table

nms <- colnames(topFeatures)

nms[which(nms == "FDR")] <- "adj.P.Val"

colnames(topFeatures) <- nms

}

total = nrow(topFeatures)

all\_topFeatures <- topFeatures #If this is not used in visualization part, erase it out.

# Note, rowname of topFeatures must be entrez ID

anot.id <- rownames(topFeatures)

JQ\_genes <- read.csv("JQ\_genes.csv", header = T, as.is = T, na.strings = "NA")

hit.inx <- match(anot.id, JQ\_genes[, "GeneID"])

gene.anot <- JQ\_genes[hit.inx, c("GeneID", "symbol", "name")]

na.inx <- is.na(hit.inx)

gene.anot[na.inx, "symbol"] <- anot.id[na.inx]

gene.anot[na.inx, "name"] <- 'NA'

write.csv(cbind(EntrezID=anot.id, signif(topFeatures,5), Symbols = gene.anot$symbol, Name=gene.anot$name), row.names=F, file=paste(filename, "\_all\_", ls, ".csv", sep=""))

## GetSigGenes (update result based on cutoffs)

# select based on p-value

hit.inx.p <- topFeatures$adj.P.Val <= p.lvl

topFeatures<-topFeatures[hit.inx.p,,drop=F]

# now rank by logFC, note, the logFC for each comparisons

# are returned in topFeatures before the AveExpr columns

# for two-class, only one column, multiple columns can be involved

# for > comparisons - in this case, use the largest logFC among all comparisons

# further filter by logFC

if (m == 1){

hit.inx <- which(colnames(topFeatures) == "AveExpr")

} else if (m == 2){

hit.inx <- which(colnames(topFeatures) == "baseMean")

} else {

hit.inx <- which(colnames(topFeatures) == "logCPM")

}

#logFC1 <- hit.inx - 2

logFC2 <<- hit.inx - 1

logfc.mat <- topFeatures[,1:logFC2, drop=F]

pos.mat <- abs(logfc.mat)

fc.vec <- apply(pos.mat, 1, max)

hit.inx.fc <- fc.vec >= fc.lvl

topFeatures <-topFeatures[hit.inx.fc,,drop=F]

#topFeatures\_max <-topFeatures[hit.inx.fc, ?

#topFeatures\_logFC1 <-topFeatures[,c(1,logFC1), drop=F]

#topFeatures\_logFC2 <-topFeatures[,c(1,logFC2), drop=F]

# Note, rowname of topFeatures must be entrez ID

anot.id <- rownames(topFeatures)

JQ\_genes <- read.csv("JQ\_genes.csv", header = T, as.is = T, na.strings = "NA")

hit.inx <- match(anot.id, JQ\_genes[, "GeneID"])

gene.anot <- JQ\_genes[hit.inx, c("GeneID", "symbol", "name")]

na.inx <- is.na(hit.inx)

gene.anot[na.inx, "symbol"] <- anot.id[na.inx]

gene.anot[na.inx, "name"] <- 'NA'

write.csv(cbind(EntrezID=anot.id, signif(topFeatures,5), Symbols = gene.anot$symbol, Name=gene.anot$name), row.names=F, file=paste(filename, "\_DEG\_", ls, ".csv", sep=""))

de.Num <- nrow(topFeatures)

return(paste("total", total, "genes,", de.Num,"differentially expressed genes", sep = " "))

}

```

# Declare functions for heatmap, MA Plot, and volcano plot

```{r}

# Heatmap

PlotHeatmap <- function(resTable){

library(RColorBrewer)

library(pheatmap)

if (nrow(resTable) > 0) {

hm <- as.matrix(resTable[,1:logFC2])

col <- colorRampPalette(rev(brewer.pal(10, "RdBu")))(256)

pheatmap(hm, show\_rownames = F, cluster\_cols = F, angle\_col = 0, border\_color=NA, col=col)

} else {

print("There is no DEG!")

}

}

# MA plot

plotma <- function(logcpm, adjp, logfc, avgexp = NULL, fc.lvl = 1, p.lvl = 0.05){

if(is.null(avgexp)){ #EdgeR

dt <- as.data.frame(cbind(logcpm = logcpm, adjp = adjp, logfc = logfc))

}else{ #Limma

dt <- as.data.frame(cbind(logcpm = logcpm, adjp = adjp, logfc = logfc, avgexp = avgexp))

}

dt$sig <- ifelse(dt$logfc > fc.lvl & dt$adjp < p.lvl, 1,

ifelse(dt$logfc < -fc.lvl & dt$adjp < p.lvl, -1, 0)

)

dt$sig <- factor(dt$sig)

if(is.null(avgexp)){ #EdgeR

maplot = ggplot(dt, aes(x= logcpm, y= logfc, color= sig))

}else{ #Limma

maplot = ggplot(dt, aes(x= avgexp, y= logfc, color= sig))

}

maplot <- maplot +

geom\_point(size=0.5, alpha=0.5) +

geom\_hline(color = "blue3", yintercept = 0) +

stat\_smooth(se = FALSE, method = "loess", color = "red3") +

scale\_color\_manual(values=c("-1" = "green", "0" = "black", "1" = "red"))+ theme(legend.position="none")

print(maplot)

}

#volcano plot

PlotVolcano <- function(logfc, adjp, fc.lvl, p.lvl){

col.inx <- ifelse(logfc > fc.lvl & adjp < p.lvl, 3,

ifelse(logfc < -fc.lvl & adjp < p.lvl, 1, 2))

MyPalette <- c(rgb(t(col2rgb("blue")), alpha=80, maxColorValue=255), rgb(t(col2rgb("black")), alpha=40, maxColorValue=255), rgb(t(col2rgb("red")), alpha=80, maxColorValue=255))

ggplot(data=cbind(logfc, -log10(adjp)), aes(x=logfc, y=-log10(adjp))) +

geom\_point(alpha=0.4, size=1.5, color = MyPalette[col.inx]) +

theme(legend.position = "none") +

xlim(c(-10, 10)) + ylim(c(0, 15)) +

xlab("log2 fold change") + ylab("-log10 p-value")

}

```

# Perform differential analysis and draw plots using previously declared functions

```{r}

# Select comparison method

# pair-wise: all pair-wise comparison (A-B) + (B-C) + (A-C)

# time-series: compare consecutive groups (B-A) + (C-B)

# custom: only compare two groups (A-C)

# reference: all others against common reference (A-C) + (B-C)

# nested: (A-B)+(C-D)

n <- readline(prompt="Enter an integer to select a comparison method (1. pair-wise 2. time-series 3. custom 4. reference): ") #select 3 or 4

n <- as.numeric(n)

m <- readline(prompt="Enter an integer to select a statistical method (1. limma 2. deseq2 3. edger): ")

m <- as.numeric(m)

# deseq2 is unavailable now.

p.lvl <- readline(prompt="Enter the threshold of p-value : ") #usually 0.05

p.lvl <- as.numeric(p.lvl)

fc.lvl <- readline(prompt="Enter the threshold of log fold change : ") #usually 1

fc.lvl <- as.numeric(fc.lvl)

# ELS

cls.lbl <- grp.lbl

cls.nms <- grp.nms

edger\_df <- df\_dup\_fixed\_filt

ls <- "ELS"

# ELS DEA

DEAnal(df\_dup\_final, n, m, p.lvl, fc.lvl)

# ELS

resTable <- read.csv(paste(filename, "\_DEG\_", ls, ".csv", sep=""), header = T, comment.char = "", check.names=F, row.names = 1, as.is = T, na.strings = "NA")

resTable\_all <- read.csv(paste(filename, "\_all\_", ls, ".csv", sep=""), header = T, comment.char = "", check.names=F, row.names = 1, as.is = T, na.strings = "NA")

logcpm <- resTable\_all[,logFC2+1] # Limma "AveExpr", DESeq2 "baseMean", EdgeR "logCPM"

adjp <- resTable\_all$adj.P.Val

logfc <- resTable\_all[,1]

#if (nrow(resTable) > 0) {HM <- as.matrix(resTable[,1:logFC2])}

DEG\_ELS <- df\_dup\_final[,match(rownames(resTable), colnames(df\_dup\_final))] #subset DEGs from count table

colnames(DEG\_ELS) <- resTable$Symbols[match(rownames(resTable), colnames(DEG\_ELS))]

fc\_ELS <- DEG\_ELS[6:15,]/colMeans(DEG\_ELS[1:5,])

fc\_ELS[fc\_ELS <= 0] <- min(fc\_ELS[fc\_ELS > 0])/10

fc\_ELS <- log2(fc\_ELS) # NaNs produced

# ELS plotting

PlotHeatmap(resTable)

plotma(logcpm, adjp, logfc)

PlotVolcano(logfc, adjp, fc.lvl, p.lvl)

if (logFC2 == 2) {

logfc <- resTable\_all[,2]

plotma(logcpm, adjp, logfc)

PlotVolcano(logfc, adjp, fc.lvl, p.lvl)

}

# adult

cls.lbl <- grp.lbl2

cls.nms <- grp.nms2

edger\_df <- df\_dup\_fixed\_filt2

ls <- "adult"

# adult DEA

DEAnal(df\_dup\_final2, n, m, p.lvl, fc.lvl)

# adult

resTable <- read.csv(paste(filename, "\_DEG\_", ls, ".csv", sep=""), header = T, comment.char = "", check.names=F, row.names = 1, as.is = T, na.strings = "NA")

resTable\_all <- read.csv(paste(filename, "\_all\_", ls, ".csv", sep=""), header = T, comment.char = "", check.names=F, row.names = 1, as.is = T, na.strings = "NA")

logcpm <- resTable\_all[,logFC2+1] # Limma "AveExpr", DESeq2 "baseMean", EdgeR "logCPM"

adjp <- resTable\_all$adj.P.Val

logfc <- resTable\_all[,1]

#if (nrow(resTable) > 0) {HM <- merge(HM, as.matrix(resTable[,1:logFC2]), all = TRUE, by = 0)}

#rownames(HM) <- HM$Row.names

#HM <- as.matrix(HM[,-1])

#colnames(HM) <- c("ELS.High-Control", "ELS.Medium-Control", "Adult.High-Control", "Adult.Medium-Control")

#HM[is.na(HM)] <- as.double("NA")

DEG\_adult <- df\_dup\_final2[,match(rownames(resTable), colnames(df\_dup\_final2))] #subset DEGs from count table

colnames(DEG\_adult) <- resTable$Symbols[match(rownames(resTable), colnames(DEG\_adult))]

fc\_adult <- DEG\_adult[6:15,]/colMeans(DEG\_adult[1:5,])

fc\_adult[fc\_adult <= 0] <- min(fc\_adult[fc\_adult > 0])/10

fc\_adult <- log2(fc\_adult)

# adult plotting

PlotHeatmap(resTable) # “group-wise” Heatmap plotting for DEGs of adult (2 groups; medium vs control, high vs control)

plotma(logcpm, adjp, logfc)

PlotVolcano(logfc, adjp, fc.lvl, p.lvl)

if (logFC2 == 2) {

logfc <- resTable\_all[,2]

plotma(logcpm, adjp, logfc)

PlotVolcano(logfc, adjp, fc.lvl, p.lvl)

}

#pheatmap(HM2, show\_rownames = F, cluster\_row = T, cluster\_cols = F, angle\_col = 0, border\_color=NA,

# na\_col = "black", col=col,

# main = paste0("Significant genes: adj. p-val < ", p.lvl, ", abs(logFC) > ", fc.lvl), scale = "none")

# “Sample-wise” heatmap plotting for DEGs of ELS and adult # top-50 by adj.P.Val

# (15 samples; 5 each for 3 groups- control, medium, high)

col <- colorRampPalette(rev(brewer.pal(10, "RdBu")))(256)

pheatmap(t(DEG\_ELS), show\_rownames = F, cluster\_cols = F, cluster\_rows = T, border\_color=NA, col=col, scale="row", main = "1 els")

pheatmap(t(DEG\_adult), show\_rownames = F, cluster\_cols = F, cluster\_rows = T, border\_color=NA, col=col, scale="row", main = "1 adult")

# (10 samples; 5 each for 2 groups- medium vs control, high vs control)

pheatmap(t(fc\_ELS), show\_rownames = F, cluster\_cols = F, cluster\_rows = T, border\_color=NA, col=col, scale="none", main = "2 fc els wo scale")

pheatmap(t(fc\_adult), show\_rownames = F, cluster\_cols = F, cluster\_rows = T, border\_color=NA, col=col, scale="none", mian = "2 fc adult wo scale")

# “Sample-wise” heatmap plotting for “common DEGs” between ELS and adult # clustered by row(genes)

# (15 samples; 5 each for 3 groups- control, medium, high)

DEG\_adult\_joint <- DEG\_adult[,colnames(DEG\_adult) %in% colnames(DEG\_ELS)]

DEG\_ELS\_joint <- DEG\_ELS[,colnames(DEG\_ELS) %in% colnames(DEG\_adult)]

pheatmap(t(rbind(DEG\_ELS\_joint, DEG\_adult\_joint)), cluster\_col = F, cluster\_r = T, border\_color=NA, col=col, scale="none", main = "3 wo scale")

#30 samples - ELS-C, M, H; Adult - C, M, H

b <- t(rbind(DEG\_ELS\_joint, DEG\_adult\_joint))

pheatmap(cbind(apply(b[,1:5],1,mean),apply(b[,6:10],1,mean),apply(b[,11:15],1,mean),apply(b[,16:20],1,mean), apply(b[,21:25],1,mean), apply(b[,26:30],1,mean)), cluster\_col = F, cluster\_r = T, border\_color=NA, col=col, scale="none", main = "4 wo scale")

# (10 samples; 5 each for 2 groups- medium vs control, high vs control)

fc\_adult\_joint <- fc\_adult[,colnames(fc\_adult) %in% colnames(fc\_ELS)]

fc\_ELS\_joint <- fc\_ELS[,colnames(fc\_ELS) %in% colnames(fc\_adult)]

pheatmap(t(rbind(fc\_ELS\_joint, fc\_adult\_joint)), cluster\_col = F, cluster\_r = T, border\_color=NA, col=col, scale="none", main = "5 wo scale")

a <- t(rbind(fc\_ELS\_joint, fc\_adult\_joint))

pheatmap(cbind(apply(a[,1:5],1,mean),apply(a[,6:10],1,mean),apply(a[,11:15],1,mean),apply(a[,16:20],1,mean)), cluster\_col = F, cluster\_r = T, border\_color=NA, col=col, scale="row", main = "6")

fc\_combine <- cbind(apply(a[,1:5],1,mean),apply(a[,6:10],1,mean),apply(a[,11:15],1,mean),apply(a[,16:20],1,mean))

colnames(fc\_combine) <- c("FC\_ELS\_medium","FC\_ELS\_high", "FC\_Adult\_medium", "FC\_Adult\_high" )

pheatmap(cbind(apply(a[,1:10],1,mean),apply(a[,11:20],1,mean)), cluster\_col = F, cluster\_r = T, border\_color=NA, col=col, scale="none", main = "7-1")

pheatmap(t(rbind(fc\_ELS\_joint, fc\_adult\_joint)), cluster\_col = F, cluster\_r = T, border\_color=NA, col=col, scale="row", main = "7-2")

```

```{r}

#Parametric and non-parametric (binf HW2)

Anova - done

t-test (independent or paired) - done

mann whitney U (independent) - done

wilcoxon (related) - done

Multiple testing corrections (bonferroni or FDR) - done

https://en.wikipedia.org/wiki/Mann%E2%80%93Whitney\_U\_test

It is possible that parametric and nonparametric outcomes are different. The data might follow normal distribution, and if we know that, then it is better to use the parametric testing.

```{r}

anov\_dat <- cbind(grp.lbl, df\_dup\_final)

aof <- function(x){

anova(aov(x ~ anov\_dat[,1]))

}

anova.res <- apply(df\_dup\_final, 2, aof)

f\_val <- c()

for(i in 1:length(anova.res)){

f\_val[i] <- anova.res[i][[1]][[4]][1]

}

head(f\_val)

p\_val <- c()

for(i in 1:length(anova.res)){

p\_val[i] <- anova.res[i][[1]][[5]][1]

}

head(p\_val)

bonf <- c()

bonf <- p.adjust(p\_val, method = "bonferroni")

head(bonf)

fdr <- c()

fdr <- p.adjust(p\_val, method = "BH")

head(fdr)

combined <- cbind(F\_val = f\_val, P\_val = p\_val,

Bonferroni = bonf, FDR = fdr)

dim(combined)

rownames(combined) <- 1:nrow(combined)

combined

pheatmap(t(df\_dup\_final[,order(combined[,2], decreasing = F)[1:200]]))

#one <- aov(data ~ grp.lbl, data = as.data.frame(df\_dup\_final))

#summary(one, test = "Wilks")

#summary.aov(one, test = "Wilks")

#three\_manova <- manova(cbind(Distance, Mass) ~ as.factor(Colony) \* Size.class, data = ant)

# summary(three\_manova, test = "Wilks")

# summary.aov(three\_manova, test = "Wilks")

# summary.manova(three\_manova, test = "Wilks")

# anova(lm(mpg ~ disp + hp, data = mtcars))

#t.test(dat1, dat2) is the same as t.test(melt(dat1), melt(dat2))... -> if we apply t.test to the matrix, they are automatically changing to the one long vector!!!

#t.test on samples

t\_samples <- data.frame(matrix(ncol = 4, nrow = choose(nrow(df\_dup\_final),2)))

x <- c("T.value", "p.value", "bonferroni", "BH")

colnames(t\_samples) <- x

tt.t <- data.frame(matrix(ncol = nrow(df\_dup\_final), nrow = nrow(df\_dup\_final)))

tt.p <- data.frame(matrix(ncol = nrow(df\_dup\_final), nrow = nrow(df\_dup\_final)))

tt.name <- data.frame(matrix(ncol = nrow(df\_dup\_final), nrow = nrow(df\_dup\_final)))

#sample pairwise comparisons forloop

for(i in 1:(nrow(df\_dup\_final)-1)){

for(j in (i+1):nrow(df\_dup\_final)){

tt.t[i,j] <- t.test(df\_dup\_final[i,], df\_dup\_final[j,], paired = F)$statistic #cuz they are all independent samples!

tt.p[i,j] <- round(t.test(df\_dup\_final[i,], df\_dup\_final[j,], paired = F)$p.value, 4)

tt.name[i,j] <- paste0(i, "+", j)

}

}

rownames(t\_samples) <- na.omit(melt(t(tt.name))$value)#Extracting triangle values row by row

t\_samples[,1] <- na.omit(melt(t(tt.t))$value)

t\_samples[,2] <- na.omit(melt(t(tt.p))$value)

bonf <- c()

bonf <- p.adjust(t\_samples$p.value, method = "bonferroni")

fdr <- c()

fdr <- p.adjust(t\_samples$p.value, method = "BH")

t\_samples[,3] <- bonf

t\_samples[,4] <- fdr

t\_samples

t\_samples[order(t\_samples$p.value),] #based on small p-values: meaning that those two samples are not the same based on t-stattistics

t\_samples[order(t\_samples$p.value, decreasing = T),] #based on small p-values: meaning that those two samples are the same based on t-stattistics

#MW U test on samples

MWU\_samples <- data.frame(matrix(ncol = 4, nrow = choose(nrow(df\_dup\_final),2)))

x <- c("W.value", "p.value", "bonferroni", "BH")

colnames(MWU\_samples) <- x

mwu.w <- data.frame(matrix(ncol = nrow(df\_dup\_final), nrow = nrow(df\_dup\_final)))

mwu.p <- data.frame(matrix(ncol = nrow(df\_dup\_final), nrow = nrow(df\_dup\_final)))

mwu.name <- data.frame(matrix(ncol = nrow(df\_dup\_final), nrow = nrow(df\_dup\_final)))

#sample pairwise comparisons forloop

for(i in 1:(nrow(df\_dup\_final)-1)){

for(j in (i+1):nrow(df\_dup\_final)){

mwu.w[i,j] <- wilcox.test(df\_dup\_final[i,], df\_dup\_final[j,], paired = F)$statistic #cuz they are all independent samples!

mwu.p[i,j] <- round(wilcox.test(df\_dup\_final[i,], df\_dup\_final[j,], paired = F)$p.value, 4)

mwu.name[i,j] <- paste0(i, "+", j)

}

}

rownames(MWU\_samples) <- na.omit(melt(t(mwu.name))$value)#Extracting triangle values row by row

MWU\_samples[,1] <- na.omit(melt(t(mwu.w))$value)

MWU\_samples[,2] <- na.omit(melt(t(mwu.p))$value)

bonf <- c()

bonf <- p.adjust(MWU\_samples$p.value, method = "bonferroni")

fdr <- c()

fdr <- p.adjust(MWU\_samples$p.value, method = "BH")

MWU\_samples[,3] <- bonf

MWU\_samples[,4] <- fdr

MWU\_samples

MWU\_samples[order(MWU\_samples$p.value),] #based on small p-values: meaning that those two samples are not the same based on MWU

MWU\_samples[order(MWU\_samples$p.value, decreasing = T),] #based on small p-values: meaning that those two samples are the same based on MWU

#t.test on genes - too much to do... choose(ncol(df\_dup\_final),2)

#MW U test on genes

```

#Unsupervised

PCA

sparse PCA

sparse independent PCA - http://mixomics.org/methods/ipca/

Canonical discrimant analysis

hierarchical clustering (try different distance matrix)

K-means clustering (try different distance matrix)

model-based clustering (try different distance matrix)

t-sne

```{r}

#PCA

pca <- prcomp(df\_dup\_final, scale = T)

#head(pca$rotation[,1:4], 30) #first four loadings

eigen1 <- (pca$sdev)^2 #eigenvalues

eigen\_data <- matrix(0, nrow = round(sum(eigen1),0), ncol = 3)

if (length(eigen1) < sum(eigen1)){

eigen1 <- c(eigen1, rep(0, sum(eigen1) - length(eigen1) + 1)) #Sometimes delete 1

}

colnames(eigen\_data) <- c("eigenvalue", "percentage", "cumulative.percentage")

rownames(eigen\_data) <- paste0("comp", 1:sum(eigen1))

eigen\_data[,1] <- round(eigen1, 4)

percentage <- apply(as.matrix(eigen1), 2, sum(eigen1), FUN = "/") \* 100

eigen\_data[,2] <- round(percentage, 4)

cum\_fun <- function(x){ #x should be n \* 1 column matrix

for (i in 2:nrow(x)){

x[i,] <- x[i-1,] + x[i,]

}

return(x)

}

cumulative <- cum\_fun(percentage) #or use cumsum!!!

eigen\_data[,3] <- cumulative

print(head(eigen\_data, 100))

barplot(eigen\_data[,1], main = "Bar-chart of eigenvalues", names = c(paste("PC", 1:sum(eigen1))),

cex.names = 0.5)

ggplot(as.data.frame(pca$x[,1:2]), aes(x = PC1, y = PC2)) + geom\_point() +

geom\_text(aes(label = rownames(pca$x), hjust = -0.4), size = 3) + ggtitle("PC plot")

plot(PCA(df\_dup\_final))

comp\_comb <- pca$x[,1:2]

comp\_comb <- cbind(comp\_comb, 0)

colnames(comp\_comb)[3] <- "Group"

comp\_comb <- as.data.frame(comp\_comb)

comp\_comb$Group <- grp.lbl

graph1 <- ggplot(comp\_comb, aes(x = PC1, y = PC2, colour = Group))

graph1 <- graph1 + geom\_point() + labs(title = "PC plots of Genes") +

geom\_vline(xintercept = 0) + geom\_hline(yintercept = 0) +

geom\_text(aes(label = rownames(comp\_comb), col = Group, hjust = -0.3), size = 2)

graph1

#Quality of individuals representation

options(scipen=999) #remove scientific number

quality <- function(pca, standardized){

new <- data.frame(matrix(NA, ncol=ncol(pca), nrow=nrow(pca)))

for (i in 1: nrow(pca)){

new[i,] <- pca[i,]^2 / sum((standardized[i,])^2)

}

return(new)

}

quality\_mat <- quality(pca$x, scale(df\_dup\_final))

round(quality\_mat, 4)

rowSums(quality\_mat) #check whether the sum of row is 1

paste("Best represented gene on the first PC:", rownames(df\_dup\_final)[which.max(quality\_mat[,1])])

paste("Best represented gene on the second PC:", rownames(df\_dup\_final)[which.max(quality\_mat[,2])])

paste("Worst represented gene on the first PC:", rownames(df\_dup\_final)[which.min(quality\_mat[,1])])

paste("Worst represented gene on the second PC:", rownames(df\_dup\_final)[which.min(quality\_mat[,2])])

fviz\_pca\_ind(pca, axes = c(1,2), #simply changed it to change the component

col.ind = "cos2", # Color by the quality of representation

gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),

repel = TRUE # Avoid text overlapping

)

#contributions of the individuals to each extracted PC

options(scipen=999)

contribution <- function(pca, eigen){

new2 <- data.frame(matrix(NA, ncol=ncol(pca), nrow=nrow(pca)))

for (i in 1:ncol(pca)){

new2[,i] <- (pca[,i]^2 / (eigen)[i]) \* 100 / (nrow(pca) - 1)

}

return(new2)

}

contribution\_mat <- contribution(pca$x, eigen1)

colSums(contribution\_mat) #Check whether the sum of column is 100; as you can see, the PC12 would give wrong answer, as the eigen value is extremely small... IT means PC12 is not valuable... (barely containing any variation)

contribution\_mat

par(mfrow = c(2,3))

for (i in 1:dim(contribution\_mat)[2]){

plot(contribution\_mat[,i], xlab = "observation", ylab = paste("contributions on PC", i))

abline(h = 100/(nrow(pca$x) - 1), lty = 1, col = "red")

}

#Print out the 90 percentile of the contribution for each PC

for (i in 1:12){

paste("PC", i,":",round(quantile(contribution\_mat[,i], 0.9), 7))

}

print("So, any of points given above numbers for each PC can be regarded as outliers")

print("and it is better to take them out, as they have influences.")

print("Influential genes are below:")

for (i in 1:ncol(contribution\_mat)){

for (j in 1:nrow(contribution\_mat)){

if(round(quantile(contribution\_mat[,i], 0.90), 7) < contribution\_mat[j,i]){

print(paste("Influential genes for PC", i," are", rownames(df\_dup\_final)[j]))

}

}

}

pcarank <- data.frame(PC1 = pca$x[,1])

pcarank$Rank <- rank(pcarank$PC1)

order <- pcarank[order(pcarank$Rank), ]

order$name <- rownames(order)

order$name <- factor(order$name, levels = order$name[order(order$PC1)])

order

ggplot(order, aes(x = name, y = PC1)) + geom\_bar(stat = "identity") +

theme(text = element\_text(size=8), axis.text.x = element\_text(angle = 40, hjust = 1)) +

ggtitle("PC")

biplot(pca$x, x)

biplot(pca, scale = 0.3)

```

```{r}

#sPCA

ns <- readline(prompt="Enter an integer for the number of components: ")

ns <- as.numeric(ns) #So, run PCA first, and then, see how many components you want to keep - so sparse PCA (sPCA) is quite helpful after you run PCA and get some idea of how many compoenents you want to keep

spca.result <- spca(df\_dup\_final, ncomp = ns, center = T, scale = T,

keepX = rep(ncol(df\_dup\_final) \* 0.5, ns)) #KeepX = #of values for each componenet

selectVar(spca.result, comp = 1) #comp can be 1 ~ ns

spcas <- spca.result$x

plotVar(spca.result, comp = 1:2, pch = 20)

plotIndiv(spca.result, group= grp.lbl, legend = TRUE, title = 'Genes')

#sIPCA

#Same idea for sIPCA...

ni <- readline(prompt="Enter an integer for the number of components: ")

ni <- as.numeric(ni)

sipca.result <- sipca(df\_dup\_final, ncomp = ni, mode = "deflation",

scale = T, keepX = rep(ncol(df\_dup\_final) \* 0.5, ns))

plotIndiv(sipca.result, group= grp.lbl, legend = TRUE, title = 'Genes')

selectVar(sipca.result, comp = 1)#comp can be 1 ~ ni

plotVar(sipca.result, comp = 1:2, pch = 20)

sipcas <- sipca.result$x

```

```{r}

#canonical discriminannt analysis

library(dummies)

y <- grp.lbl #class

numeric.y <- as.numeric(y)

x <- pca$x[,1:5] #design matrix (but we can use pca, ipca, sipca = the reason why i did not use original data was since the # of columns is bigger than the # of rows, so it will cause singularity issue when we try to use solve function later. But i get that this is not the best idea, cause we will lose interpretation, but for now, as what we have this is the best decision i could make) - can change "1:5"

c <- matrix(0, dim(x)[2], length(levels(y)))

getC <- function(x, y, c){

combined <- as.data.frame(cbind(y = y, x = x))

combined$y <- as.factor(combined$y)

splited <- split(combined, combined$y)

xjbar <- apply(x, 2, mean)

n <- nrow(combined)

for(i in 1:length(levels(combined$y))){

coef <- nrow(splited[[i]]) / (n - 1)

for(j in 1:dim(x)[2]){

xkjbar <- mean(splited[[i]][,j + 1])

c[j,i] <- sqrt(coef) \* (xkjbar - xjbar[j])

}

}

return(c)

}

C <- getC(x, numeric.y, c)

C

within\_variance <- function(predictors, response){

y <- dummy(response)

x <- scale(predictors, T, F)

x <- as.matrix(x)

n <- nrow(predictors)

w <- 1/(n-1) \* t(x) %\*% (diag(n) - y %\*% solve(t(y) %\*% y) %\*% t(y)) %\*% x

return(w)

}

between\_variance <- function(predictors, response){

y <- dummy(response)

x <- scale(predictors, T, F)

x <- as.matrix(x)

n <- nrow(predictors)

b <- 1/(n-1) \* t(x) %\*% y %\*% solve(t(y) %\*% y) %\*% t(y) %\*% x

return(b)

}

total\_variance <- function(predictors){

center <- scale(predictors, T, F)

n <- nrow(predictors)

v <- 1/(n-1) \* t(center) %\*% center

return(v)

}

W <- within\_variance(x, numeric.y)

B <- between\_variance(x, numeric.y)

Total <- total\_variance(numeric.y)

w <- eigen(t(C) %\*% solve(W) %\*% C)$vectors

eigen(t(C) %\*% solve(W) %\*% C)$values

u <- solve(W) %\*% C %\*% w

z <- as.matrix(x) %\*% u

z <- as.data.frame(z)

z$type <- grp.lbl

z\_len <- ncol(z) - 1

ggplot(data = z, aes(x = V1, y = V2, col = type)) + geom\_point()

ggplot(data = z, aes(x = V1, y = V3, col = type)) + geom\_point()

ggplot(data = z, aes(x = V2, y = V3, col = type)) + geom\_point()

cor(z[,-ncol(z)], df\_dup\_final) #correlation of CDA and our data

#=========================Compare with PCA==============================

library(amap)

pca2 <- acp(df\_dup\_final)

score2 <- as.data.frame(pca2$scores)

score2$class <- grp.lbl

ggplot(score2, aes(x = `Comp 1`, y = `Comp 2`, col = class)) + geom\_point() #Seems like CDA with PCA can outperform PCA itself only

#==========================================================

#Create a matrix of size n × K, with the squared Mahalanobis distances d2(xi, gk) of each observation xi (i.e. each sample) to the each of the k centroids gk. The squared distance, with the Mahalanobis metric.

#So, it would be way better if we calculate the distance with the original data matrix,, but since we have more columns... we cannot...

x <- as.data.frame(pca$x[,1:5])

mah <- matrix(0, nrow(x), z\_len)

pure.len <- ncol(x)

x$class <- grp.lbl

splited3 <- split(x, x$class)

for(j in 1:z\_len){

gk <- apply(splited3[[j]][,-(pure.len+1)], 2, mean)

for(i in 1:nrow(x)){

factor <- as.matrix(x[i, -(pure.len+1)] - gk)

mah[i, j] <- factor %\*% solve(W) %\*% t(factor)

}

}

#assign each observation to the class Gk for which the Mahalanobis distance d2(xi, gk) is the smallest. And create a confussion matrix comparing the actual class versus the predicted class. (20 pts)

assign <- data.frame(observation = paste("observation",1:nrow(mah)), group = 0)

for(i in 1:nrow(mah)){

assign[i,2] <- which.min(mah[i, ])

}

assign

assign[,2] == as.numeric(grp.lbl)

assign$actual <- grp.lbl

table(assign$group, assign$actual)

```

```{r}

#Cluster

#c. model based

library(mclust)

fit <- Mclust(df\_dup\_final)

plot(mclustBIC(df\_dup\_final))

plot(fit, what = "classification")

plot(fit, what = "uncertainty")

plot(fit, what = "density")

summary(fit)

cat("This is how they are classified:", fit$classification)

cat("The matrix of probability for each observation is:")

round(fit$z, 8) #number of columns is the same as the # of clusters

cat("The uncertainties for each observation is:", fit$uncertainty)

fit$BIC #same as mclustBIC(univ[,-1])

paste("The optimal BIC value is", round(fit$bic, 5),

"and the optimal number of mixture components (clusters) are",

fit$G, "and the model where the optimal BIC occurs (best covariance structure) is",

fit$modelName)

summary(fit, parameters = T, classification = T)

plot(fit, what = "BIC", dimens = c(2, 5))

plot(fit, what = "classification", dimens = c(2, 5))

plot(fit, what = "uncertainty", dimens = c(2, 5))

plot(df\_dup\_final[,c(3,6)], col = fit$classification) #I just randomly picked two genes...

points(t(fit$parameters$mean[c(2,5),]), col = 1:3, pch = 8, cex = 2)

#Since there are so many genes, i think when we do model based cluster, its better for us to select a few genes, and perform it...

i <- 3 #again, I just randomly picked two genes...

j <- 6

plot(df\_dup\_final[,i], df\_dup\_final[,j], xlab = colnames(df\_dup\_final)[i], ylab = colnames(df\_dup\_final)[j], type="n")

for(k in 1:nrow(df\_dup\_final)){

text(df\_dup\_final[k,i], df\_dup\_final[k,j], grp.lbl, col = fit$classification[k])

}

points(t(fit$parameters$mean[c(2,5),]), col = 1:3, pch = 8, cex = 2)

#b. kmeans

set.seed(100)

kmean <- kmeans(df\_dup\_final, fit$G) #Apply k-means to this data with the number of clusters equal to the best number found above...

kmean

summary(kmean)

plot(df\_dup\_final[,c(3,6)], col = kmean$cluster)

points((kmean$centers[,c(2,5)]), col = 1:3, pch = 8, cex = 2) #Repeat the same thing with clusters found by kmeans now

i <- 3

j <- 6

plot(df\_dup\_final[,i], df\_dup\_final[,j], xlab = colnames(df\_dup\_final)[i], ylab = colnames(df\_dup\_final)[j], type="n")

for(k in 1:nrow(df\_dup\_final)){

text(df\_dup\_final[k,i], df\_dup\_final[k,j], grp.lbl, col = kmean$cluster[k])

}

points((kmean$centers[,c(2,5)]), col = 1:3, pch = 8, cex = 2)

#a. hirerarchial

hc1<-hclust(dist(df\_dup\_final), "average")

plot(hc1)

plot(hc1, hang=-1, main="Hierarchial cluster: avg", ylab=NULL)

hc2<-hclust(dist(df\_dup\_final), "complete")

plot(hc2)

plot(hc2, hang=-1, main="Hierarchial cluster: complete", ylab=NULL)

hc3<-hclust(dist(df\_dup\_final), "single")

plot(hc3)

plot(hc3, hang=-1, main="Hierarchial cluster: single", ylab=NULL)

library(dplyr)

hierach <- df\_dup\_final %>% dist(method = "euclidean") %>% hclust(method = "average") #method = complete, single

hierach

summary(hierach)

plot(hierach, hang = -1, main = "Hierarchial; avg, euclidean", ylab = NULL)

rect.hclust(hierach, k = fit$G, border="red") #use the fit that we got in the previous method for dividing into boxes

grouping <- cutree(hierach, k = fit$G) #use the fit that we got in the previous method for dividing into boxes

print("Here is a clustering results:")

print(grouping)

#One of the problems of K-means algorithm is that they need to define K before the algorithm runs, but hierarchial clustering does not need to define k beforehand. In hierarchial clustering, we need to define n-clusters (where n is the number of obervations), and then, generally merge from bottom to up, until there is only one cluster left. So, we are basically repeatedly combining the two clusters with the shortest distance each other. And, there are different types of cluster dissimilarity measures (linkage).

hc.complete <- df\_dup\_final %>% dist(method = "euclidean") %>% hclust(method = "complete")

hc.average <- df\_dup\_final %>% dist(method = "euclidean") %>% hclust(method = "average")

hc.single <- df\_dup\_final %>% dist(method = "euclidean") %>% hclust(method = "single")

par(mfrow = c(1,3))

plot(hc.complete)

rect.hclust(hc.complete, k = 3, border="red")

plot(hc.average)

rect.hclust(hc.average, k = 3, border="red")

plot(hc.single)

rect.hclust(hc.single, k = 3, border="red")

a <- cutree(hc.complete, k = 3) #make confusion matrix to compare the methods how well they agree each other

b <- cutree(hc.average, k = 3)

c <- cutree(hc.single, k = 3)

table(a, b)

table(a, c)

table(b, c)

```

```{r}

#t-SNE

#https://www.r-bloggers.com/playing-with-dimensions-from-clustering-pca-t-sne-to-carl-sagan/

library(Rtsne)

library(grDevices)

colors = rainbow(length(unique(grp.lbl)))

names(colors) = unique(grp.lbl)

par(mgp=c(2.5,1,0))

tsne <- Rtsne(df\_dup\_final, dims = 2, perplexity=4, verbose=T, max\_iter = 500)

plot(tsne$Y, t='n', main="tsne for RNAseq JQ", cex.main=2, cex.lab=1.5)

text(tsne$Y, labels=grp.lbl, col=colors[grp.lbl]) #col=colors()[as.numeric(grp.lbl) + 40]

tsne\_plot <- function(dims=3, perpl=4,iterations=500,learning=200){

set.seed(2019)

tsne <- Rtsne(df\_dup\_final, dims = dims, perplexity=perpl, verbose=T, max\_iter=iterations, eta=learning)

plot(tsne$Y, t='n', main = print(paste0("Dim = ",dims, ", perplexity = ",perpl,

", max\_iter = ",iterations, ", learning rate = ",learning)),

xlab="tSNE dimension 1",

ylab="tSNE dimension 2", cex.main=1, cex.lab=1.5)

text(tsne$Y, labels=grp.lbl, col=colors[grp.lbl])

}

perplexity\_values <- c(1,2,3,4)

sapply(perplexity\_values,function(i){tsne\_plot(perpl=i)})

iteration\_values <- c(10,50,100,1000)

sapply(iteration\_values,function(i){tsne\_plot(iterations=i)})

learning\_values <- c(20,200,2000)

sapply(learning\_values,function(i){tsne\_plot(learning=i)})

dim\_values <- c(2,3)

sapply(perplexity\_values,function(i){tsne\_plot(dims=i)})

d\_tsne\_1 <- as.data.frame(cbind(tsne$Y, label = 0))

d\_tsne\_1$label <- grp.lbl

ggplot(d\_tsne\_1, aes(x=V1, y=V2, col = label)) +

geom\_point(size=1) +

guides(colour=guide\_legend(override.aes=list(size=8))) +

xlab("") + ylab("") +

ggtitle("t-SNE") +

theme\_light(base\_size=10) +

theme(axis.text.x=element\_blank(),

axis.text.y=element\_blank())

#Compare with other methods

d\_tsne\_1\_original <- d\_tsne\_1 #tsne

fit\_cluster\_kmeans <- kmeans(scale(d\_tsne\_1[,-(ncol(d\_tsne\_1))]), length(levels(grp.lbl))) #kmeans cluster

d\_tsne\_1\_original$cl\_kmeans = factor(fit\_cluster\_kmeans$cluster)

fit\_cluster\_hierarchical=hclust(dist(scale(d\_tsne\_1[,-(ncol(d\_tsne\_1))]))) #hierarchial cluster

d\_tsne\_1\_original$cl\_hierarchical = factor(cutree(fit\_cluster\_hierarchical, k=length(levels(grp.lbl))))

plot\_cluster <- function(data, var\_cluster, palette){

ggplot(data, aes\_string(x="V1", y="V2", color=var\_cluster)) +

geom\_point(size=1) +

guides(colour=guide\_legend(override.aes=list(size=6))) +

xlab("") + ylab("") +

ggtitle("") +

theme\_light(base\_size=10) +

theme(axis.text.x=element\_blank(),

axis.text.y=element\_blank(),

legend.direction = "horizontal",

legend.position = "bottom",

legend.box = "horizontal") +

scale\_colour\_brewer(palette = palette)

}

plot\_k <- plot\_cluster(d\_tsne\_1\_original, "cl\_kmeans", "Accent")

plot\_h <- plot\_cluster(d\_tsne\_1\_original, "cl\_hierarchical", "Set1")

grid.arrange(plot\_k, plot\_h, ncol=2) #Seems like kmeans and hierarchial made the same clusters...

```

#Extra

Enrichment analysis - gene set enrichment analysis from enrichR package (ecotoxx)

multi-omics analysis. (i.e. Two-way orthogonal partial least square analysis, regularized CCA, adjusted RV, etc.) - (o2pls, mixomics)

```{r}

#rCCA

#Classical CCA assumes that p < n and q < n, where p and q are the number of variables in each set. In the high dimensional setting usually encountered with biological data, where p + q >> n + 1, CCA cannot be performed

grid1 <- seq(0.01, 0.2, length = 5)

grid2 <- seq(0.01, 0.2, length = 5)

#Its so slow... so i cut the dimensions... this is demo so we can change it later.

cv <- tune.rcc(df\_dup\_final[,1:700], df\_dup\_final2[,1:300], grid1 = grid1, grid2 = grid2, validation = "loo")

result <- rcc(df\_dup\_final[,1:700], df\_dup\_final2[,1:300], ncomp = 3, lambda1 = cv$opt.lambda1, lambda2 = cv$opt.lambda2)

plot(result, scree.type = "barplot")

round(result$cor, 4)

plotIndiv(result, comp = 1:2,

group = grp.lbl, rep.space = "XY-variate",

legend = TRUE, title = 'rCCA')

plotIndiv(result, comp = 1:2,

group = grp.lbl2,

legend = TRUE, title = 'rCCA')

result$variates#These are the canonical variates for the two data sets - it is analous to PCs from PCA.

```

```{r}

#PLS & sPLS

library(mixOmics)

#PLS

result.pls <- pls(df\_dup\_final[,1:1000], df\_dup\_final2[,1:400], ncomp = 10) # where ncomp is the number of dimensions/components to choose

tune.pls <- perf(result.pls, validation = 'loo', criterion = 'all', progressBar = FALSE) #this is to find how many components to keep

plot(tune.pls$Q2.total) #seems like i can only keep 2 components- can use R2 or MSEP

abline(h = 0.0975)

plotIndiv(result.pls, comp = 1:2, rep.space= 'Y-variate', group = grp.lbl,

legend = TRUE, title = 'PLS comp 1 annd 2 Block Y')

plotIndiv(result.pls, comp = 1:2, rep.space= 'X-variate', group = grp.lbl,

legend = TRUE, title = 'PLS comp 1 annd 2 Block X')

plotIndiv(result.pls, comp = 1:2, rep.space= 'XY-variate', group = grp.lbl,

legend = TRUE, title = 'PLS comp 1 annd 2 Block XY')

plotIndiv(result.pls, comp = 1:2, rep.space= 'Y-variate', group = grp.lbl2,

legend = TRUE, title = 'PLS comp 1 annd 2 Block Y')

plotIndiv(result.pls, comp = 1:2, rep.space= 'X-variate', group = grp.lbl2,

legend = TRUE, title = 'PLS comp 1 annd 2 Block X')

plotIndiv(result.pls, comp = 1:2, rep.space= 'XY-variate', group = grp.lb2,

legend = TRUE, title = 'PLS comp 1 annd 2 Block XY')

col.tox <- color.mixo(as.numeric(grp.lbl))

plotIndiv(result.pls, ind.names = F, axes.box = "both", col = col.tox, style = '3d')

plotVar(result.pls, comp =1:2, cex = c(4, 5))

# define red and green colors for the edges

color.edge <- color.GreenRed(50)

# to save as a pdf

network(result.pls, comp = 1:2, shape.node = c("rectangle", "rectangle"),

color.node = c("white", "pink"), color.edge = color.edge, threshold = 0.8)

cim(result.pls, comp = 1:3, margins = c(7, 7))

# SPLS

ncomp = 10

result.spls <- spls(df\_dup\_final[,1:1000], df\_dup\_final2[,1:400], ncomp = ncomp, keepX = c(rep(10, ncomp)), mode = 'invariant')

tune.spls <- perf(result.spls, validation = 'Mfold', folds = 10,

criterion = 'all', progressBar = FALSE)

plot(tune.spls$Q2.total) #seems like i can only keep 1 component- can use R2 or MSEP

abline(h = 0.0975)

plotIndiv(result.spls, comp = 1:2, rep.space= 'Y-variate', group = grp.lbl,

legend = TRUE, title = 'sPLS comp 1 annd 2 Block Y')

plotIndiv(result.spls, comp = 1:2, rep.space= 'X-variate', group = grp.lbl,

legend = TRUE, title = 'sPLS comp 1 annd 2 Block X')

plotIndiv(result.spls, comp = 1:2, rep.space= 'XY-variate', group = grp.lbl,

legend = TRUE, title = 'sPLS comp 1 annd 2 Block XY')

plotIndiv(result.spls, comp = 1:2, rep.space= 'Y-variate', group = grp.lbl2,

legend = TRUE, title = 'sPLS comp 1 annd 2 Block Y')

plotIndiv(result.spls, comp = 1:2, rep.space= 'X-variate', group = grp.lbl2,

legend = TRUE, title = 'sPLS comp 1 annd 2 Block X')

plotIndiv(result.spls, comp = 1:2, rep.space= 'XY-variate', group = grp.lb2,

legend = TRUE, title = 'sPLS comp 1 annd 2 Block XY')

col.tox <- color.mixo(as.numeric(grp.lbl))

plotIndiv(result.spls, ind.names = F, axes.box = "both", col = col.tox, style = '3d')

plotVar(result.spls, comp = 1:2, cex = c(4, 5))

# define red and green colors for the edges

color.edge <- color.GreenRed(50)

# to save as a pdf

network(result.spls, comp = 1:2, shape.node = c("rectangle", "rectangle"),

color.node = c("white", "pink"), color.edge = color.edge, threshold = 0.8)

cim(result.spls, comp = 1:3, margins = c(7, 7))

```

```{r}

#o2pls

set.seed(2019)

library(OmicsPLS)

rownames(DEG\_adult) <- rownames(DEG\_ELS)

table\_cv <- crossval\_o2m\_adjR2(DEG\_adult, DEG\_ELS,

a = 1:6,

ax = 0:2,

ay = 0:2, nr\_folds = 4, nr\_cores = 4)

table\_cv <- table\_cv[order(table\_cv$MSE), ]

table\_cv <- head(table\_cv)#I can show them to the users

for(i in 1:5){ #output the best optimal 5 models based on MSE

name <- paste0("fit", i)

assign(name, o2m(df\_dup\_final, df\_dup\_final2, table\_cv$n[i], table\_cv$nx[i], table\_cv$ny[i]))

}

table\_cv

summary\_variance <- data.frame(matrix(ncol = 14, nrow = 5))

x <- c("n", "nx", "ny", "X - noise", "Y - noise", "X\_joint", "Y\_joint", "X\_orth", "Y\_orth",

"X\_by\_Y", "Y\_by\_X",

"X\_joint\_by\_Y\_joint", "Y\_joint\_by\_X\_joint", "MSE")

colnames(summary\_variance) <- x

#This definitely needs to be shown!!!

for(i in 1:5){ #output the best optimal 5 models based on MSE

name <- paste0("fit", i)

assign(name, o2m(df\_dup\_final, df\_dup\_final2, table\_cv$n[i], table\_cv$nx[i], table\_cv$ny[i]))

getting <- get(name)

vec <- c(table\_cv$n[i],

table\_cv$nx[i],

table\_cv$ny[i],

round(getting$R2X \* 100, 3),

round(getting$R2Y \* 100, 3),

round(getting$R2Xcorr \* 100, 3),

round(getting$R2Ycorr \* 100, 3),

round(getting$R2X\_YO \* 100, 3),

round(getting$R2Y\_XO \* 100, 3),

round(getting$R2Xhat \* 100, 3),

round(getting$R2Yhat \* 100, 3),

round(getting$R2Xhat/getting$R2Xcorr \* 100, 3),

round(getting$R2Yhat/getting$R2Ycorr \* 100, 3),

table\_cv$MSE[i]

)

summary\_variance[i,] <- vec

}

summary\_variance

fit <- fit5 #change this later

#variance table2 & SSQ plots (joint, ortho, noise) & o2pls model joint summary [in the paper Stacey Reinke "OnPLS xxx"]

summary\_score\_noise <- data.frame(matrix(ncol = 6, nrow = 2))

x <- c("X\_joint\_score", "X\_ortho\_score",

"Y\_joint\_score", "Y\_ortho\_score",

"X\_noise", "Y\_noise"

)

colnames(summary\_score\_noise) <- x

rownames(summary\_score\_noise) <- c("Absolute", "Relative")

summary\_score\_noise[1,] <- c(sum(summary(fit)[12]$flags$varXjoint),

sum(summary(fit)[12]$flags$varXorth),

sum(summary(fit)[12]$flags$varYjoint),

sum(summary(fit)[12]$flags$varYorth),

(summary(fit)[12]$flags$ssqX - sum(summary(fit)[12]$flags$varXjoint) - sum(summary(fit)[12]$flags$varXorth)),

(summary(fit)[12]$flags$ssqY - sum(summary(fit)[12]$flags$varYjoint) - sum(summary(fit)[12]$flags$varYorth))

)

summary\_score\_noise[2,] <- c(round(fit$R2Xcorr \* 100, 3),

(round(fit$R2X \* 100, 3) - round(fit$R2Xcorr \* 100, 3)),

round(fit$R2Ycorr \* 100, 3),

(round(fit$R2Y \* 100, 3) - round(fit$R2Ycorr \* 100, 3)),

(100 - round(fit$R2X \* 100, 3)),

(100 - round(fit$R2Y \* 100, 3))

)

summary\_score\_noise

model\_summary <- data.frame(matrix(ncol = 3, nrow = summary(fit)[12]$flags$n + 1))

x <- c("component", "ELS", "Adult")

colnames(model\_summary) <- x

y <- c(1:summary(fit)[12]$flags$n, "sum")

model\_summary[,1] <- y

first <- c()

second <- c()

for (i in 1:summary(fit)[12]$flags$n){

first[i] <- summary(fit)[12]$flags$varXjoint[i]/summary(fit)[12]$flags$ssqX

second[i] <- summary(fit)[12]$flags$varYjoint[i]/summary(fit)[12]$flags$ssqY

}

first <- c(first, sum(first))

second <- c(second, sum(second))

model\_summary[,2] <- round(first, 4) \* 100

model\_summary[,3] <- round(second, 4) \* 100

model\_summary

#3. SSQ

xsq <- rbind(apply((fit$Tt %\*% t(fit$W.))^2, 2, sum),

apply((fit$T\_Yosc %\*% t(fit$P\_Yosc.))^2, 2, sum),

apply((fit$E)^2, 2, sum))

rownames(xsq) <- c("joint", "orth", "noise")

xsq <- t(xsq)

sum\_xsq <- apply(xsq, 1, sum)

xsq <- melt(xsq)

xsq$Var2 <- factor(xsq$Var2, labels = c("joint", "orth", "noise"))

ggplot(xsq, aes(x = Var1, y = value, fill = Var2)) + geom\_bar(stat = "identity") +

labs(title = "ELS: SSQ per variable", y = "SSQ", x = "Variable") +

theme(axis.text.x=element\_blank(),

axis.ticks.x=element\_blank()) +

guides(fill=guide\_legend(title="Types"))

ggplot(xsq, aes(x = Var1, y = value, fill = Var2)) + geom\_bar(stat = "identity", position = "fill") +

labs(title = "ELS: SSQ per variable (normalized)", y = "SSQ", x = "Variable") +

theme(axis.text.x=element\_blank(),

axis.ticks.x=element\_blank()) +

guides(fill=guide\_legend(title="Types"))

melt(sort(sum\_xsq, decreasing = T)[1:10])

ysq <- rbind(apply((fit$U %\*% t(fit$C.))^2, 2, sum),

apply((fit$U\_Xosc %\*% t(fit$P\_Xosc.))^2, 2, sum),

apply((fit$Ff)^2, 2, sum))

rownames(ysq) <- c("joint", "orth", "noise")

ysq <- t(ysq)

sum\_ysq <- apply(ysq, 1, sum)

ysq <- melt(ysq)

ysq$Var2 <- factor(ysq$Var2, labels = c("joint", "orth", "noise"))

ggplot(ysq, aes(x = Var1, y = value, fill = Var2)) + geom\_bar(stat = "identity") +

labs(title = "Adult: SSQ per variable", y = "SSQ", x = "Variable") +

theme(axis.text.x=element\_blank(),

axis.ticks.x=element\_blank()) +

guides(fill=guide\_legend(title="Types"))

ggplot(ysq, aes(x = Var1, y = value, fill = Var2)) + geom\_bar(stat = "identity", position = "fill") +

labs(title = "Adult: SSQ per variable (normalized)", y = "SSQ", x = "Variable") +

theme(axis.text.x=element\_blank(),

axis.ticks.x=element\_blank()) +

guides(fill=guide\_legend(title="Types"))

melt(sort(sum\_ysq, decreasing = T)[1:10])

#First joint loadings of X

load1x <- rbind(fit$W.[,1], fit$P\_Yosc.[,1])

rownames(load1x) <- c("joint", "orth")

load1x <- t(load1x)

abssum\_load1x <- apply(abs(load1x), 1, sum)

melt(sort(abssum\_load1x, decreasing = T)[1:10])

melt(sort(abs(load1x[,1]), decreasing = T)[1:10])

melt(sort(abs(load1x[,2]), decreasing = T)[1:10])

load1x <- melt(load1x)

load1x$Var2 <- factor(load1x$Var2, labels = c("joint", "orth"))

ggplot(load1x, aes(x = Var1, y = value, fill = Var2)) + geom\_bar(stat = "identity") +

labs(title = "ELS: Loadings", y = "Loading value", x = "Variable") +

theme(axis.text.x=element\_blank(),

axis.ticks.x=element\_blank()) +

guides(fill=guide\_legend(title="Types"))

#First joint loadings of Y

load1y <- rbind(fit$C.[,1], fit$P\_Xosc.[,1])

rownames(load1y) <- c("joint", "orth")

load1y <- t(load1y)

abssum\_load1y <- apply(abs(load1y), 1, sum)

melt(sort(abssum\_load1y, decreasing = T)[1:10])

melt(sort(abs(load1y[,1]), decreasing = T)[1:10])

melt(sort(abs(load1y[,2]), decreasing = T)[1:10])

load1y <- melt(load1y)

load1y$Var2 <- factor(load1y$Var2, labels = c("joint", "orth"))

ggplot(load1y, aes(x = Var1, y = value, fill = Var2)) + geom\_bar(stat = "identity") +

labs(title = "Adult: Loadings", y = "Loading value", x = "Variable") +

theme(axis.text.x=element\_blank(),

axis.ticks.x=element\_blank()) +

guides(fill=guide\_legend(title="Types"))

#table of variables with top loadings (joint and orth)

#Joint X

for (i in 1:dim(fit$W.)[2]){

name <- paste0("top\_joint\_x\_", i)

assign(name, data.frame(matrix(ncol = 2, nrow = 10)))

getting <- get(name)

colnames(getting) <- c("Symbol", "Loading Value")

getting[,1] <- names(head(sort(abs(fit$W.[,i]),decreasing=TRUE, index.return = T)$x, 10))

getting[,2] <- fit$W.[,i][head(sort(abs(fit$W.[,i]),decreasing=TRUE, index.return = T)$ix, 10)]

assign(name, getting)

}

#quality of predictions of X with Y

for(i in 1:dim(fit$Tt)[2]){

name <- paste0("qual", i)

assign(name, lm(fit$Tt[,i] ~ fit$U[,i]))

getting <- get(name)

print(summary(getting)[4]) #coefficients

print(summary(getting)[9]) #adj R^2

f <- summary(getting)$fstatistic

print(pf(f[1], f[2], f[3], lower.tail=FALSE)) #fstatistics p-value

par(mfrow = c(2, 2))

plot(getting, which = 1)

plot(getting, which = 4)

plot(getting, which = 5)

plot(fit$U[,i], fit$Tt[,i], main = name, xlab = paste0(name, "\_Yscore"), ylab = paste0(name, "\_Xscore"));abline(lm(fit$Tt[,i] ~ fit$U[,i]), col = "red")

}

#quality of predictions of Y with X

for(i in 1:dim(fit$U)[2]){

name <- paste0("Score", i)

assign(name, lm(fit$U[,i] ~ fit$Tt[,i]))

getting <- get(name)

print(summary(getting)[4]) #coefficients

print(summary(getting)[9]) #adj R^2

f <- summary(getting)$fstatistic

print(pf(f[1], f[2], f[3], lower.tail=FALSE)) #fstatistics p-value

par(mfrow = c(2, 2))

plot(getting, which = 1)

plot(getting, which = 4)

plot(getting, which = 5)

plot(fit$Tt[,i], fit$U[,i], main = name, xlab = paste0(name, "\_Xscore"), ylab = paste0(name, "\_Yscore"));abline(lm(fit$U[,i] ~ fit$Tt[,i]), col = "red")

}

```

#Supervised - regression and classification (maybe model selection using bic, cp, AIC, etc)

```{r}

```