Gale Lab Expression Analysis Pipeline

Leanne Whitmore

August 2020

R Markdown

This markdown outlines steps the Gale lab takes for analysing RNA-seq gene expression

```
count_matrix_path <- ""

#this file contains metadata such status of animals
#(i.e protected not protected) and other information such animal sex etc ...
target_file_path <- ""

##name of folder where results are placed
results_folder="Results"
```

Load libraries

```
library(MASS)
library(rrcov)
library(stats)
library(factoextra)
library(umap)
library(Rtsne)
library(ggplot2)
library(data.table)
library(gplots)

if (!require("edgeR")) {
  if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
    BiocManager::install("edgeR")
  library(edgeR)
}


if (!require("limma")) {
  install.packages("limma", repos="http://cran.rstudio.com/")
  library("limma")
}
```

Load in necessary functions

```
generate_folder <- function(foldername) {
    #function generates results folder

    workDir <- getwd()
    subDir <- foldername
```

```
    results_path <- file.path(workDir, subDir)
    if (file.exists(subDir)) {
    } else {
        dir.create(results_path)
    }
    return(results_path)
}

filter_read_counts <- function(countm, filter_cutoff) {
    #Filter value was calculated by row sums

    A <- rowSums(countm)
    isexpr <- A >= filter_cutoff
    cmfl <- countm[isexpr, ]
    return(cmfl)
}

generate_density_plot <- function(data, labels, filename, figres) {
    #generate density plots for counts

      png(filename, res = figres)
    par(xpd = TRUE)
    if (length(labels) > 10) {
        plotDensities(data, legend = FALSE)
    } else {
        plotDensities(data,
            legend = "topright",
            inset = c(-0.2, 0), levels(labels)
        )
    }
    dev.off()
}

vizualize_mds <- function(data, labels, results_path,
                          base_file_name, figres=100) {
    ###MDS (multidimensional scaling) uses
    ###log fold changes between genes as distances
    MDS <- plotMDS(data, gene.selection = "pairwise", cex =  .8, plot = FALSE)
    minx <- min(MDS$x)
    maxx <- max(MDS$x)
    miny <- min(MDS$y)
    maxy <- max(MDS$y)
    png(file.path(results_path,
        paste0("mds_", base_file_name)))

    plot(MDS$x, MDS$y, cex = 1, xlim = c(minx - 1, maxx + 1),
         ylim = c(miny - 1, maxy + 1),
         xlab = paste0(MDS$axislabel, " 1"),
         ylab = paste0(MDS$axislabel, " 2"), frame = FALSE)
    text(MDS$x, MDS$y, labels, cex = 0.6, pos = 4)
    dev.off()
}
```

```
pca_fun <- function(exprs, labels, results_path,
                    base_file_name, target_columns,
                    figres=100) {
    #Run PCA/SVD reduction

    pca <- prcomp(t(exprs))
    E <- get_eig(pca)
    cx <- sweep(t(exprs), 2, colMeans(t(exprs)), "-")
    sv <- svd(cx)

    vizualize_pca(file.path(results_path, paste0("svd_", base_file_name)),
                  sv$u, labels[, target_columns[1]],
                 labels[, target_columns[2]], figres, E)
    vizualize_pca(file.path(results_path, paste0("pca_", base_file_name)),
                  pca$x, labels[, target_columns[1]],
                  labels[, target_columns[2]],
                  figres, E)
    vizualize_scree_plot(file.path(results_path,
                                   paste0("scree_", base_file_name)),
                         pca, figres)

    loadingscores <- as.data.frame(pca$rotation)
    is_pc1_0 <- loadingscores$PC1 > 0
    is_pc2_0 <- loadingscores$PC2 > 0

    loadingscores <- loadingscores[is_pc1_0,]
    loadingscores <- loadingscores[with(loadingscores, order(-PC1)),]
    save_loading_scores(file.path(results_path, "loadingscores_pc1.txt"),
                        loadingscores["PC1"], figres)

    loadingscores <- as.data.frame(pca$rotation)
    loadingscores <- loadingscores[is_pc2_0, ]
    loadingscores <- loadingscores[with(loadingscores, order(-PC2)), ]
    save_loading_scores(file.path(results_path, "loadingscores_pc2.txt"),
                        loadingscores["PC2"], figres)

}

umap_fun <- function(exprs, labels, results_path,
                     base_file_name, target_columns,
                     figres=100) {
    #Runs default paramaters of umap
    U <- umap(t(exprs))
    vizualize_umap(file.path(results_path, paste0("umap_", base_file_name)),
                   U$layout, labels[, target_columns[1]],
                   labels[, target_columns[2]], figres)
}

vizualize_umap <- function(plot_file, U, class1, class2, figres) {
    #Vizualize umap reduction
    minx <- min(U[, 1])
    maxx <- max(U[, 1])
    miny <- min(U[, 2])
```

```
    maxy <- max(U[, 2])
    png(plot_file, res = figres)
    par(mar = c(5, 4, 2, 4), xpd = TRUE)
    plot(U[, 1], U[, 2], frame = FALSE,
         ylim = c(miny - 1, maxy + 1), xlim = c(minx - 1, maxx + 1),
         pch = as.numeric(as.factor(class1)),
         col = as.numeric(as.factor(class2)),
         xlab = "Dim 1", ylab = "Dim 2")
    legend("topright", inset = c(-0.25, -0.1), bty = "n",
           pch = as.numeric(levels(as.factor(as.numeric(as.factor(class1))))),
           legend = levels(as.factor(class1)))
    legend("bottomright", inset = c(-0.25, 0), bty = "n", pch = "-",
            col = levels(as.factor(as.numeric(as.factor(class2)))),
           legend = c(levels(as.factor(class2))))
    dev.off()
}

vizualize_pca <- function(plot_file, PCA, class1, class2, figres, E) {
    #Vizualize PCA  results
    minx <- min(PCA[, 1])
    maxx <- max(PCA[, 1])
    miny <- min(PCA[, 2])
    maxy <- max(PCA[, 2])
    png(plot_file, res = figres)
    par(mar = c(5, 4, 2, 5.5), xpd = TRUE)
    plot(PCA[, 1], PCA[, 2], frame = FALSE,
         ylim = c(miny, maxy), xlim = c(minx, maxx),
         pch = as.numeric(as.factor(class1)),
         col = as.numeric(as.factor(class2)),
         xlab = paste0("PC1 ", round(E$variance.percent[1],  digits = 2), "%"),
         ylab = paste0("PC2 ", round(E$variance.percent[2],  digits = 2), "%"))
    legend("topright", inset = c(-0.35, -0.1), bty = "n",
           pch = as.numeric(levels(as.factor(as.numeric(as.factor(class1))))),
           legend = levels(as.factor(class1)))
    legend("bottomright", inset = c(-0.37, 0), bty = "n", pch = "-",
           col = levels(as.factor(as.numeric(as.factor(class2)))),
           legend = c(levels(as.factor(class2))))
    dev.off()
}

vizualize_scree_plot <- function(plot_file, PCA, figres) {
    #Vizualize principle component variation results
    scree.plot <- fviz_eig(PCA, addlabels = TRUE, hjust = -0.3)
    png(plot_file, res = figres)
    print(scree.plot)
    dev.off()
}

save_loading_scores <- function(write_file, df, figres) {
    #Save list of genes that have a positive effect on variation of principle
    #component 1 and 2 sorted from most influential
    write.table(df, file = write_file)
}
```

```
tsne_fun <- function(exprs, labels, results_path,
                     base_file_name, target_columns, figres=100) {
    #Runs default paramaters of umap
    T <- Rtsne(t(exprs), perplexity = 1)

    vizualize_tSNE(file.path(results_path, paste0("tsne_", base_file_name)),
                   T$Y, labels[, target_columns[1]],
                   labels[, target_columns[2]], figres)
}
vizualize_tSNE <- function(plot_file, U, class1, class2, figres) {
    #Vizualize umap reduction
    minx <- min(U[, 1])
    maxx <- max(U[, 1])
    miny <- min(U[, 2])
    maxy <- max(U[, 2])
    png(plot_file, res = figres)
    par(mar = c(5, 4, 2, 4), xpd = TRUE)
    plot(U[, 1], U[, 2], frame = FALSE, ylim = c(miny - 1, maxy + 1),
         xlim = c(minx - 1, maxx + 1),  pch = as.numeric(as.factor(class1)),
         col = as.numeric(as.factor(class2)),
         xlab = "Dim 1", ylab = "Dim 2")
    legend("topright", inset = c(-0.25, -0.1), bty = "n",
            pch = as.numeric(levels(as.factor(as.numeric(as.factor(class1))))),
           legend = levels(as.factor(class1)))
    legend("bottomright", inset = c(-0.25, 0), bty = "n", pch = "-",
           col = levels(as.factor(as.numeric(as.factor(class2)))),
           legend = c(levels(as.factor(class2))))
    dev.off()
}
```

Begining of main analysis: load necessary files and preprocess counts

```
# load in count matrix and targetfile, make sure that sep option matches how your file is separated
# also it is important that the column header of the count matrix and row.names of the target file matc

cm <- read.table(count_matrix_path,
    header = TRUE, sep = "\t", row.names = 1,
    as.is = TRUE, check.names = FALSE, stringsAsFactors = FALSE
)

target <- read.table(target_file_path,
    sep = ",", row.names = 1, as.is = TRUE,
    check.names = FALSE, header = TRUE, stringsAsFactors = FALSE
)

generate_density_plot(
    as.matrix(cm), rownames(target),
    file.path(results_folder,
              "densities_raw_counts.png"),
    100
)
```

```
#filter out lowly expressed genes
cm <- filter_read_counts(cm, length(colnames(cm))*35)  #35 average number of read counts per gene accro

# Ensures rownames of target file are in the same order of the column names in the count matrix
cm <- cm[, rownames(target)]

# CHECK IF ORDER IS THE SAME
if (all.equal(colnames(cm), rownames(targetfile)) != TRUE) {
    print ('MASSIVE WARNING: RESULTS WILL BE WRONG IF THIS IS NOT EQUAL!!!!!!!!')
    print(rownames(targetfile))
    print(colnames(cm))
}

#generate results folder
generate_folder(results_folder)
```

Normalize counts

```
#normalizaton factors (should be columns in your target file)
factor1 <- "" #i.e time
factor2 <- "" #i.e animalID

# design matrix for normalization make sure not to include only techinical reasons for variation
ti <- factor(target[, factor1])
Xid <- factor(target[, factor2])
mm <- model.matrix(~ 0 + ti + Xid)
rownames(mm) <- colnames(cm)
colnames(mm) <- make.names(colnames(mm))
mm <- mm[, colnames(mm)[order(tolower(colnames(mm[, ])))]]
mm <- mm[, colSums(mm) > 0]

#checks to make sure design matrix is feasible
exclude0 <- nonEstimable(mm)
if ("ti" %in% exclude0) {
    return("interactions term non estimable")
}
mm <- mm[, !colnames(mm) %in% exclude0]
if (!is.fullrank(mm)) {
    return("not full rank")
}

# normalize
cm <- DGEList(counts = cm)
cm <- calcNormFactors(cm, method = "TMM") #TMM normalization
Pi.CPM <- voom(counts = cm, design = mm, normalize.method = "none", plot = F, span = 0.1)
write.csv(Pi.CPM$E, file.path(results_folder, "1.norm_matrix.csv"))
```

Evaulate full transciptome using PCA, UMAP, MDS and tSNE feature reduction algorithms

```
##Coloring can be changed in PCA, UMAP or tsne plots by changing the values of
# factor2 or factor1 values to different columns in the target file
factor1 <- "" #i.e Protection status
factor2 <- "" #i.e animalID

print("STATUS: MDS scaling")
```

```
vizualize_mds(
    Pi.CPM$E,
    rownames(target),
    results_folder, "TranscriptAnalysis.png",
    60
)

print("STATUS: Running PCA feature reduction")
pca_fun(Pi.CPM$E, target,
        results_folder, "TranscriptAnalysis.png",
        c(factor2, factor1), 100)

print("STATUS: Running UMAP feature reduction")
umap_fun(
    Pi.CPM$E, target,
    results_folder, "TranscriptAnalysis.png",
    c(factor2, factor1), 100
)


print("STATUS: Running tSNE feature reduction")
tsne_fun(Pi.CPM$E, target,
        results_folder, "TranscriptAnalysis.png",
         c(factor2, factor1), 100)
```

DE Analysis

```
##NOTE: DEPENDING ON ANALYSIS THIS SECTION MAY NEED TO BE ALTERED
factor1 <- "" #i.e time
factor2 <- "" #i.e ProtectionStatus
factor3 <- "" #i.e animalID

# design matrix for normalization and DE analysis
ti <- factor(target[, factor1])
vacc <- factor(target[, factor2])
Xid <- factor(target[, factor3])
mm_all <- model.matrix(~ 0 + ti:vacc + Xid)
rownames(mm_all) <- colnames(cm)
colnames(mm_all) <- make.names(colnames(mm_all))
mm_all <- mm_all[, colnames(mm_all)[order(tolower(colnames(mm_all[, ])))]]
mm_all <- mm_all[, colSums(mm_all) > 0]

#checks to make sure design matrix is feasible
excludeAll <- nonEstimable(mm_all)
if ("ti" %in% excludeAll) {
    return("interactions term non estimable")
}
mm_all <- mm_all[, !colnames(mm_all) %in% excludeAll]
if (!is.fullrank(mm)) {
    return("not full rank")
}

Pi.lmfit <- lmFit(Pi.CPM, design = mm_all)
```

```
#NOTE - THIS CONTRAST MATRIX WILL BE HIGHLY DEPENDENT ON
#EXPERIMENT AND WILL NEED TO BE ALTERED BY THE USER
#TO FIT THEIR EXPERIMENT (THIS IS JUST AN EXAMPLE CONTRAST MATRIX)
contrastsmatrix <- c(
    "tiWOD1.vaccProt -tiWOD0.vaccProt",
    "tiWOD3.vaccProt -tiWOD0.vaccProt",
    "tiWOD7.vaccProt -tiWOD0.vaccProt",
    "tiWOD1.vaccNotProt -tiWOD0.vaccNotProt",
    "tiWOD3.vaccNotProt -tiWOD0.vaccNotProt",
    "tiWOD7.vaccNotProt -tiWOD0.vaccNotProt"
)

contr <- makeContrasts(contrasts = contrastsmatrix, levels = mm_all)
contrast <- contrasts.fit(Pi.lmfit, contrasts = contr)
Pi.contrasts <- eBayes(contrast, robust = TRUE, trend = TRUE)

results <- decideTests(Pi.contrasts, lfc = 0.58, method = "separate", adjust.method = "BH", p.value = 0
write.csv(Pi.contrasts$coefficients, file = file.path(results_folder, "/coefficients_O&S.csv"), quote =
write.csv(Pi.contrasts$t, file = file.path(results_folder, "t_stats_O&S.csv"), quote = F)
write.csv(Pi.contrasts$p.value, file = file.path(results_folder, "p_value_O&S.csv"), quote = F)

for (i in 1:ncol(Pi.contrasts$p.value)) Pi.contrasts$p.value[, i] <- p.adjust(Pi.contrasts$p.value[, i]
write.csv(Pi.contrasts$p.value, file = file.path(results_folder, "p_value_adj_O&S.csv"), quote = F)


dataMatrix <- Pi.contrasts$coefficients
sigMask <- dataMatrix * (results**2) # 1 if significant, 0 otherwise
ExpressMatrix <- subset(dataMatrix, rowSums(sigMask) != 0)

print(paste0("STATUS: Number of DE genes ", dim(ExpressMatrix)[1]))

# filter for significant genes - up/down regulated
sigMask <- subset(sigMask, rowSums(sigMask) != 0)
Pi.contrasts$genes <- data.frame(ID_REF = rownames(Pi.contrasts))

write.csv(ExpressMatrix, file = file.path(results_folder, "expression_matrix_de.csv"), quote = F)
write.csv(results, file = file.path(results_folder, "results_de.csv"), quote = F)
write.csv(dataMatrix, file = file.path(results_folder, "full_expression_matrix_de.csv"), quote = F)
```