Swiss Institute of Bioinformatics

Isabelle Dupanloup
BCF - Bioinformatics Core Facility

Day 3 hands-on: DESEQ2 and functional analysis

Loading R packages

```
library(DESeq2)
library(pheatmap)
library(org.Mm.eg.db)
library(DOSE)
library(pathview)
library(clusterProfiler)
library(AnnotationHub)
library(ensembldb)
library(tidyverse)
```

Data description and importation

The dataset used in this practical session corresponds to 6 mouse samples at 2 different developmental stages: newborn and adult.
The dataset corresponds to 2 different files:

- Day2_dataset_genecounts.txt : contains the gene counts for each sample and for each gene
- Day2_dataset_infossamples.txt : contains the information relative to the samples

The files can be loaded in R using:

```
data <- read.table("Day3_dataset_genecounts.txt", header = TRUE, row.names = 1)
data <- as.matrix(data)
meta <- read.table("Day3_dataset_infossamples.txt", header = TRUE)
```

Count normalization using DESeq2

Match the metadata and counts data

We should always make sure that we have sample names that match between the two files, and that the samples are in the right order. DESeq2 will output an error if this is not the case.

```
### Check that sample names match in both files
all(colnames(data) %in% rownames(meta))
all(colnames(data) == rownames(meta))
rownames(meta) <- meta$labels
```

Create DESEq2 object

Let's start by creating the DESeqDataSet object.
To create the object, we will need the count matrix and the metadata table as input. We will also need to specify a design formula. The design formula specifies the column(s) in the metadata table and how they should be used in the analysis. For our dataset we only have one column we are interested in, that is ~group. This column has two factor levels, which tells DESeq2 that for each gene we want to evaluate gene expression change with respect to these different levels.

This practical has been adapted from online tutorials developed by members of the teaching team at the Harvard Chan Bioinformatics Core (HBC): https://github.com/hbctraining/DGE_workshop

```
## Create DESeq2Dataset object
dds <- DESeqDataSetFromMatrix(countData = data, colData = meta, design = ~ group)
```

You can use DESeq-specific functions to access the different slots and retrieve information, if you wish. For example, suppose we wanted the original count matrix we would use counts().

```
view(counts(dds))
```

Generate the normalized counts

To perform the median of ratios method of normalization, DESeq2 has a single estimateSizeFactors() function that will generate size factors for us. We will use the function in the example below, but in a typical RNA-seq analysis this step is automatically performed by the DESeq() function, which we will see later.

```
dds <- estimateSizeFactors(dds)
```

By assigning the results back to the dds object we are filling in the slots of the DESeqDataSet object with the appropriate information. We can take a look at the normalization factor applied to each sample using:

```
sizeFactors(dds)
```

Now, to retrieve the normalized counts matrix from dds, we use the counts() function and add the argument normalized=TRUE.

```
normalized_counts <- counts(dds, normalized=TRUE)
```

We can save this normalized data matrix to file for later use:

```
write.table(normalized_counts, file=" normalized_counts.txt", sep="\t", quote=F,
col.names=NA)
```

NOTE: DESeq2 doesn't actually use normalized counts, rather it uses the raw counts and models the normalization inside the Generalized Linear Model (GLM). These normalized counts will be useful for downstream visualization of results, but cannot be used as input to DESeq2 or any other tools that peform differential expression analysis which use the negative binomial model.

QC for DE analysis using DESeq2

Transform normalized counts using the rlog function

To improve the distances/clustering for the PCA and heirarchical clustering visualization methods, we need to moderate the variance across the mean by applying the rlog transformation to the normalized counts.

```
rld <- rlog(dds, blind=TRUE)
```

The blind=TRUE argument results in a transformation unbiased to sample condition information. When performing quality assessment, it is important to include this option.

Principal components analysis (PCA)

DESeq2 has a built-in function for plotting PCA plots, that uses ggplot2 under the hood. This is great because it saves us having to type out lines of code and having to fiddle with the different ggplot2 layers. In addition, it takes the rlog object as an input directly, hence saving us the trouble of extracting the relevant information from it.

The function plotPCA() requires two arguments as input: an rlog object and the intgroup (the column in our metadata that we are interested in).

```
plotPCA(rld, intgroup="group")
```

What does this plot tell you about the similarity of samples? Does it fit the expectation from the experimental design? By default the function uses the top 500 most variable genes. You can change this by adding the ntop argument and specifying how many genes you want to use to draw the plot.

Hierarchical Clustering

Since there is no built-in function for heatmaps in DESeq2 we will be using the pheatmap() function from the pheatmap package. This function requires a matrix/dataframe of numeric values as input, and so the first thing we need to is retrieve that information from the rld object:

```
### Extract the rlog matrix from the object
rld_mat <- assay(rld)
```

Then we need to compute the pairwise correlation values for samples.

```
rld_cor <- cor(rld_mat)
```

And now to plot the correlation values as a heatmap:

```
pheatmap(rld_cor)
```

Overall, we observe pretty high correlations across the board (> 0.999) suggesting no outlying sample(s). Also, similar to the PCA plot you see the samples clustering together by sample group. Together, these plots suggest to us that the data are of good quality and we have the green light to proceed to differential expression analysis.

Differential expression analysis with DESeq2

To run the actual differential expression analysis, just call the function DESeq().

```
dds <- DESeq(dds)
```

Let's take a look at the dispersion estimates for our dataset:

```
plotDispEsts(dds)
```

Since we have a small sample size, for many genes we see quite a bit of shrinkage.

To build our results table we will use the results() function.:

```
res <- results(dds)
```

To summarize the results table, a handy function in DESeq2 is summary().

```
summary(res)
```

In addition to the number of genes up- and down-regulated at the default threshold, the function also reports the number of genes that were tested (genes with non-zero total read count), and the number of genes not included in multiple test correction due to a low mean count.

The MA plot shows the mean of the normalized counts versus the log2 foldchanges for all genes tested. The genes that are significantly DE are colored to be easily identified.

```
plotMA(res, ylim=c(-2,2))
```

Functional analysis with clusterProfiler

Over-representation analysis with clusterProfiler

To perform the over-representation analysis, we need a list of background genes and a list of significant genes. For our background dataset we will use all genes tested for differential expression (all genes in our results table). For our significant gene list we will use genes with p-adjusted values less than 0.05 (we could include a fold change threshold too if we have many DE genes).

```
# Create background dataset for hypergeometric testing using all genes tested for
significance in the results
all_genes <- as.character(rownames(res))

# Extract significant results
signif_res <- res[res$padj < 0.05 & !is.na(res$padj), ]
signif_genes <- as.character(rownames(signif_res))
```

Now we can perform the GO enrichment analysis and save the results:

```
# Run GO enrichment analysis
ego <- enrichGO(gene = signif_genes,
                universe = all_genes,
                keyType = "ENSEMBL",
                OrgDb = org.Mm.eg.db,
                ont = "BP",
                pAdjustMethod = "BH",
                qvalueCutoff = 0.05,
                readable = TRUE)

# Output results from GO analysis to a table
cluster_summary <- data.frame(ego)
```

Visualizing clusterProfiler results

clusterProfiler has a variety of options for viewing the over-represented GO terms. We will explore the dotplot, enrichment plot, and the category netplot.

The dotplot shows the number of genes associated with the first 50 terms (size) and the p-adjusted values for these terms (color). This plot displays the top 50 genes by gene ratio (# genes related to GO term / total number of sig genes), not p-adjusted value.

```
dotplot(ego, showCategory=50)
```

The next plot is the enrichment GO plot, which shows the relationship between the top 50 most significantly enriched GO terms (padj.), by grouping similar terms together. The color represents the p-values relative to the other displayed terms (brighter red is more significant) and the size of the terms represents the number of genes that are significant from our list.

```
emapplot(ego, showCategory=50)
```

Finally, the category netplot shows the relationships between the genes associated with the top five most significant GO terms and the fold changes of the significant genes associated with these terms (color). The size of the GO terms reflects the pvalues of the terms, with the more significant terms being larger. This plot is particularly useful for hypothesis generation in identifying genes that may be important to several of the most affected processes.

```
# To color genes by log2 fold changes
signif_res_lFC <- signif_res$log2FoldChange

cnetplot(ego,
         categorySize="pvalue",
         showCategory = 5,
         foldChange= signif_res_lFC,
         vertex.label.font=6)
```

Optional: gene set enrichment analysis (GSEA) using clusterProfiler and Pathview

To perform GSEA analysis of KEGG gene sets, clusterProfiler requires the genes to be identified using Entrez IDs for all genes in our results dataset. We also need to remove the NA values and duplicates (due to gene ID conversion) prior to the analysis:

```
mart <- useDataset("mmusculus_gene_ensembl", useMart("ensembl"))
genes <- getBM(filters="ensembl_gene_id",
         attributes=c("ensembl_gene_id", "entrezgene_id"),
         values= all_genes,
         mart=mart)

indNA = which(is.na(genes$entrezgene_id))
genes_noNA <- genes[-indNA,]
indnodup = which(duplicated(genes_noNA$ entrezgene_id) == F)
genes_noNA_nodup <- genes_noNA[indnodup,]

lFC <- res$log2FoldChange[-indNA]
lFC <- lFC[indnodup]
names(lFC) <- genes_noNA_nodup$entrezgene_id

# Sort fold changes in decreasing order
lFC <- sort(lFC, decreasing = TRUE)
```

Perform the GSEA using KEGG gene sets:

```
gseaKEGG <- gseKEGG(geneList = lFC,
            organism = "mmu",
            nPerm = 1000, # default number permutations
            minGSSize = 5, # minimum gene set size
            pvalueCutoff = 0.1, # padj cutoff value
            verbose = FALSE)

# Extract the GSEA results
gseaKEGG_results <- gseaKEGG@result
```