Quantitative Life Sciences
December 7, 2022

Marius Zeeb, Marco Labarile
marius.zeeb@usz.ch

RNAseq analysis in R
Exercise sheet

Part 1: RNAseq data manual

Introduction

In this exercise we will analyze data taken from a typical experiment in medical science. Our example data is taken from a RNAseq experiment. You can read about the source of the data on: Verma S., Du P., et. al. (2018). Tuberculosis in advanced HIV infection is associated with increased expression of IFN and its downstream targets. BMC Infectious Diseases 18:220. doi: 10.1186/s12879-018-3127-4

To do the analysis you need to know that

(1) the experiment contains totally 31 samples from people with HIV

(2) 16 samples from patients with tuberculosis and 15 without

(3) there are 25369 variables measured on each sample

The experiment can in other words be summarized as 16 + 15 columns of samples times 25369 rows of variables. We want to find out which, if any, of these 25369 variables are changing (in medical science we often say "expressed" or "regulated") significantly as a result of tuberculosis infection.

In RNAseq data the variables measured are usually called "probes", "genes" or "features", and each measured value is often called an "expression" value. Don't get stuck on these terms during the exercise.

RNAseq

In medical science we often ask questions like: "How does this medicine affect the liver", "What is the difference between normal skin and a tumor in the skin", or "Is there a good candidate molecule in a tissue with inflammation that we can target with a medicine to reduce the inflammation". A common denominator is that we want to compare two or more cases, and the best experiment is naturally one that gives as much information as possible on what is going on.

More or less everything that is "going on" in a tissue or a cell in the body is controlled by proteins. Proteins are complex molecules that come in thousands of brands. Some of them are building blocks of the cell, others transmit information or store energy, and another group, the enzymes, are the work-horses that perform the actions and chemical reactions in the cell. A cell only properly functions if all these proteins are present in the correct concentrations.

Almost any change that a cell experiences is reflected in increasing and/or decreasing concentrations of one or more of these proteins. In other words, if we have a method to compare all the protein concentrations in the healthy and sick cells or tissues, we could pinpoint which are affected. This could give clues to what has actually happened in the sick sample, especially if we are lucky to know something about the affected proteins. Measuring proteins turns out to be very difficult, but instead of the proteins we can measure mRNAs: An mRNA is another kind of molecule which is used to construct the proteins. Every protein is built using one specific kind of mRNA, and the more of that mRNA we have, the more protein is produced.

Here is where the RNAseq comes in. RNAseq uses next generation sequencing (NGS) to sequence all sequences in the sample. After assembling the reads, we classify them to their corresponding gene, responsible for the particular mRNA, via databases which store sequence- and gene information. In addition we can count the number of unique reads to get the expression-level of each gene.

RNAseq and microarrays, a less flexible measurement method, have been used extensively in medical science during the last 20 years.

Installing packages

BioConductor is a project that aims to develop and make available R functionality for bioinformatics, that is, the computational analysis of biological problems. It's a kind of umbrella that collects R libraries (also called packages) from people and research groups all over the world. BioConductor has a peer review process that each package has to undergo in order to get into the project. You can read everything about BioConductor on its web page, www.bioconductor.org.

In order to install BioConductor packages, you use a procedure that is a bit different from the usual install.packages() or R CMD INSTALL (or whatever you use for standard R packages). For the prompts use "a" or "yes" (whatever fits).

!!IF problems arise with library installations below of part 2, you may use the file "starter.ipynb" for Part 2!!

You can run this file on google.colab ("https://colab.research.google.com/") via a browser (will require a google account). You should start this now (if you have errors below) since it may take a while to install everything.

Part 2 libraries

```
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
##Install Packages
BiocManager::install("clusterProfiler", version = "3.16")
BiocManager::install("pathview")
BiocManager::install("enrichplot")
BiocManager::install("AnnotationDbi")
BiocManager::install(organism, character.only = TRUE)
install.packages(ggridges)
install.packages(matrixStats)
install.packages(tidyverse)
```

Part 1

```
BiocManager::install("TBSignatureProfiler")
install.packages(tidyverse)
install.packages(matrixStats)
install.packages(limma)
```

Activate the packages in the normal way:

```
library(organism, character.only = TRUE)
library(clusterProfiler)
library("org.Hs.eg.db")
library(enrichplot)
library(GenomeInfoDb)
library(org.Hs.eg.db)
```

```
library(TBSignatureProfiler)
library(SummarizedExperiment)
library(pathview)
library(ggridges)
library(limma)
library(matrixStats)
library(tidyverse)
library(AnnotationDbi)
```

Retrieving data

The data is automatically loaded when you load the "TBSignatureProfiler" package. Access it via the following line and extract the data matrix with counts:

```
hivtb_data = TB_hiv
counts = hivtb_data@assays@data@listData[["counts"]]
```

You can view the data if you do:

```
View(counts)
```

You see the matrix with the 25369 variables/genes in the rows and the different samples (patients) represented by the columns. The names of the genes (rownames) are in the "GeneCards Symbol"-format, more to that later. Columnnames with starting with "R01" are TB positive. The actual values are the counts of mRNA sequence reads corresponding t that particular gene.

Question 1: Check the distribution of the samples via a boxplot.

```
...(counts)
```

This looks strange! Almost all values are close to zero and there seems to be an extended tail of outlying high values. Actually, RNAseq data is not at all normal distributed. But if we apply the 10-logarithm to the data it looks much better:

```
logcounts =
```

Check again if it improved:

```
boxplot(logcounts)
```

This looks much better. Furthermore, the mean of all the samples are approximately the same, showing that on a global scale, RNAseq worked equally well for all samples technically.

In a meaningful experiment with many variables, the purpose is to detect the (relatively small) subset of variables that change significantly between the conditions studied. Naively, we could now just make a lot of t-tests, one for each probe and see which are significantly changing. But this will give us a lot of false positives just by chance, even if there is no true change of any probe between the samples. (more to that later)

Question 2 (no programming required): How many false positives would you expect when investigating changes in 22000 probes between identical samples at a p-value threshold of 0.05?

Thus, we want to eliminate as many variables as possible that we do not consider biologically meaningful before doing statistical testing. Furthermore in our kind of experiment with RNAseq, data tend to be very noisy when the signal is very low.

Question 3: Let's have a look at that. Plot the standard deviation against the mean for every probe.

It is clear that standard deviation is higher at lower means. Some probes have very high standard deviations - these may be truly changing, something we will investigate further on.

Let's first eliminate the 25% of the probes that have weakest average signal:

Hint: You may find the function rowMeans() helpful.

```
q25 = quantile(probemeans, 0.25)
whichtosave = which(probemeans > q25)
q25logdata = logcounts[whichtosave,]
```

Since we are only interested in probes that change, and thus have high variability, we can also remove those with very low variability. A way to do that is to filter on the inter-quartile range (using the IQR function). Keep only those with an IQR above 1:

```
mydata = q25logdata[apply(q25logdata, 1, IQR) > 1, ]
```

Question 4: How many variables remain?

Data exploration

At this point it would be interesting to do a Principal Component Analysis (PCA). This basically transforms the data to find new orthogonal variables that explain most of the variation in the dataset, but with fewer variables than the original ones.

This variation can be analysed either between probes (rows) or samples (columns). We will use the function prcomp() for the PCA. It does the analysis between rows. For our purpose the samples are most interesting to compare, so we need to interchange columns and rows in mydata. This is also called to "transpose" the matrix.

```
tdata = t(mydata)
```

Now we use function prcomp() on the transposed data:

```
pca = prcomp(tdata, scale=T)
```

We can look at the explanatory value of the principal components:

```
summary(pca)
```

The first component explains the largest part of the variance, but not all. In the best of worlds, this accounts for the difference between our experimental conditions (TB vs non TB), otherwise we have some unknown batch effect that dominate the experiment.

We can plot the samples in relation to the first two components:

```
plot(pca$x, type="n")
text(pca$x, rownames(pca$x), cex=0.5)
```

More interesting is maybe to see which experimental condition (TB infection in our case) they belong to.

```
conditions = c(rep("TB",16),rep("NOTB",15)
plot(pca$x, type="n")
text(pca$x, labels=conditions, cex=0.5)
```

The first component divides the samples by condition. However, far from perfectly. Some samples with TB are clustering with the non TB samples. That can be worth remembering when judging the final results.

Another informative plot is to do a dendrogram of correlation between the samples. First we create a correlation matrix between the samples, then a hierarchical clustering is performed:

```
pearsonCorr = as.dist(1 - cor(mydata))
hC = hclust(pearsonCorr)#,method='average')
plot(hC, labels = conditions)
```

The heights of the branches indicate how distant the samples are, i.e., how different are the gene expression levels.

Compare this to the publication (Figure 1a)(tuberculosis_transcriptomics_paper.pdf) Why do you think it looks different?

The mixture of the different samples with different TB status is even more evident here than in the PCA.

Another useful visualization of large datasets is the heatmap. R clusters the data both on rows columns with this command:

```
heatmap(mydata)
```

Red corresponds to high, yellow to low values. The dendrogram on top is basically the same as we produced earlier, in a slightly different order. Towards the bottom of the heatmap are the probes clustered that discriminate the two conditions clearly.

Let's now find the probes that differ significantly between TB status. There are several tools to do that, more or less advanced. The most simplistic would be to do a t-test for every probe. This is however not a good idea. We have a lot of probes, and the few samples will give the estimate of variance low precision in many cases and give us many false negatives and positives. However, it turns out that the variance of probes with approximately the same expression level is rather similar, and hence one can let probes "borrow" variance from each other to get better variance estimates. Such a method is employed in the limma package (Linear Models for Microarrays). limma needs to see the whole dataset, including the high variance probes, to do correct variance estimations. Thus we will go back and use our dataset containing all data. In addition to the actual data, limma needs a model matrix, basically information on which conditions each sample represents. R has a standard function for defining model matrices, model.matrix. It uses the tilde operator (the wiggly line) to define dependencies. Each input variable should be a factor, so let's first make a factor out of the conditions:

```
condfactor = factor(conditions)
```

Construct a model matrix and assign the names to the columns:

```
design = model.matrix(~0+condfactor)
```

For the columns of the design matrix you could use any names. Lets choose "noTB" for the control samples, and "TB" for TB infected.

```
colnames(design) = c("noTB", "TB")
```

Check how the matrix looks:

```
design
```

Note that the first 16 samples have '1' in the TB level, and the following 15 have the '1' in the noTB level. If we had had a multifactor experiment (ANOVA style data), we would have included more levels and assigned them in appropriate combinations to the samples. The next command estimates the variances:

```
fit = lmFit(mydata, design)
```

Now we need to define the conditions we want to compare - trivial in this case since there are only two conditions:

```
contrastmatrix = makeContrasts(TB - noTB,levels=design)
```

The following commands calculate the p-values for the differences between the conditions defined by the contrast matrix:

```
fit = contrasts.fit(fit, contrastmatrix)
ebayes = eBayes(fit)
```

As so often in R we can use show(ebayes) or just ebayes to see what the result contains:

```
ebayes
```

A lot of stuff here, one of the most interesting is the p.value. Let's make a histogram of the p values:

```
hist(ebayes$p.value)
```

You see that the number of probes are enriched close to $p = 0.00$. This surplus is due to the genes that are significantly changing between TB status. If the data had a skewed distribution we might see an accumulation at the $p = 1.00$ end, indicating a problem with our data.

If the data were completely random, the p values would be equally distributed from 0 to 1, thus if we from random data picked the probes that had $p < 0.05$ as "significant", we would pick exactly 1/20 of all probes, all being false positives. Unfortunately, there is no way to discriminate the surplus true probes from the false positives. This problem exists in any study where a lot of variables are tested, for instance in large sociology studies. However, there are ways to regulate the p-value cut off in order to have control over the false positives. A common way is the Benjamini-Hochberg adjustment. This adjustment allows you to set a limit to how large fraction of false positive variables (false discovery rate, or FDR) you accept in the results.

This adjustment, at 5% FDR, is built into the decideTests function:

```
results = decideTests(ebayes)
```

decideTests produces 0, 1, or -1 for each probes, telling if that probe did not pass the test (0), or was significantly increased (1), or decreased (-1)

Extract the original data for the most changing probes:

```
resData =mydata[results != 0,]
```

Add gene symbols as row names:

```
geneSymbol = as.array(rownames((mydata))
gs = geneSymbol[c(which(results != 0))]
rownames(resData) = gs
```

Add p-values in an extra column:

```
pvalues = ebayes$p.value[results != 0,]
resData = cbind(resData, pvalues)
```

And add-p values corrected for multiple testing (q-values):

```
adj.pvalues = p.adjust(ebayes$p.value, method="BH")
adj.pvalues = adj.pvalues[results != 0]
resData = cbind(resData, adj.pvalues)
```

Write output to a file:

```
write.table(resData, "most_regulated.txt", sep="\t")
```

Now we have generated both quality control graphs and a table of probes that change significantly between the samples. The gene names are mostly rather cryptic, feel free to search the web or continue with part 2 for "more interpretable" results.

Part 2: Analysis with predefined pipelines from R packages

As an alternative to the manual approach of part 1 we can use prebuild packages from a R to run the analysis. There isn't much coding to do in Part 2. It rather is meant to show "state-of-the-art" RNAseq methodology and for biological exploration of the results. There is no right or wrong here! If you run into errors (dependecy issues), you may use the browser based starter script as mentioned above.

Finding significantly differently expressed genes

For this we will use the package "edgeR". The following code is loosely taken from their example code adapted to our data .(https://bioconductor.org/packages/release/bioc/vignettes/edgeR/inst/doc/edgeRUsersGuide.pdf)

First load the data again and also extract the counts per million.

```
hivtb_data = TB_hiv
hivtb_data = mkAssay(hivtb_data, log = TRUE, counts_to_CPM = TRUE)
```

Now look for genes with at least three (you can play with this values) counts above 0.5 counts per million in each group (noTB/TB). You can access the gene names and the data with the following:

```
hivtb_data@NAMES
hivtb_data@assays@data@listData[["counts_cpm"]])
```

Extract now the raw gene counts and filter those out with less than sample genes with 0.5 counts per million, transform all samples columns into numeric variables. Use the code below as the basis.

```
countdat = cbind(hivtb_data@NAMES,hivtb_data@assays@data@listData[["counts"]])
```

Now create the edgeR object. (group "1" = TB, group "0" = noTB)

```
egr = DGEList(counts = as.matrix(countdat[,2:32]), group = c(rep(1,16),rep(0,15)))
```

The following function normalizes the measurements between samples

```
egr = calcNormFactors(egr)
```

After that we need to calculate the dispersion, i.e., the distribution the of gene counts. This will be done in two ways following each other.

1. The common dispersion, i.e., assuming equal mean and standard deviation among genes.

2. Tagwise dispersion, i.e., assuming gene-specific dispersions (or of genes with a similar distribution). We can also plot those dispersion parameters.

```
egr = estimateDisp(egr)
plotBCV(egr)
```

Now we are already ready to run statistical test to distinguish between TB status on all genes. The test is based on a negative binomial distribution.

```
et = exactTest(egr)
```

Finally we can plot these results (so called volcano plot). Each dot represents a gene, the x axis is the log2fold change (comparing patients with TB to patients without TB), and the y axis is the negative log10 of the p-value.

```
et$table %>%
  mutate(logp = -log10(PValue)) %>%
  ggplot(., aes(logFC,logp )) +
  geom_point(aes(color = logp), size = 2/5) +
  xlab(expression("log2 fold change")) +
  ylab(expression("-log10 pvalue")) +
  scale_color_viridis_c()
```

Question : Compare the plot to Figure 1b and discuss it with your neighbour. Do you see a similar trend?

Gene enrichment analysis

Now that we have a long list of cryptic genes and how they are similar/different based on TB status. We can use gene enrichment analysis to compress it a bit into more "readable", summarized groups.

Use the provided code in the starter_script. This code is loosely taken from "https://learn.gencore.bio.nyu.edu/rna-seq-analysis/gene-set-enrichment-analysis/" and adapted to our data.

Question 1 Use the help function or any search engine to learn about the parameters of the gseGO-function and adjust them as you see fit. E.g., change the pAdjustMethod-paramter to "fdr".

Question 2 Can you make sense of the two different example plots?

Question 3 Since our analysis is quite simplified we might not find the same results as the original paper. Can you still find similarities? Compare your findings to the ones from the original publication table 2. (tuberculosis_transcriptomics_paper.pdf)

Bonus: KEGG Pathway analysis

KEGG (Kyoto Encyclopedia of Genes and Genomes) is a database which stores information about genes and their related biological pathways. We can use this to visualize the genes in our dataset in their respective pathway.

Use the code in starter_script to visualize pathways and overlay them with our corresponding data. Loosely taken and adapted from "https://learn.gencore.bio.nyu.edu/rna-seq-analysis/gene-set-enrichment-analysis/".

Question 1 As with the gseGO-function you can change the parameters of the gseKEGG-functions.

Question 2 Look at pathways which are heavily up-/down regulated or any of your interest. Is anything surprising/unexpected?