RNA-Seq Data Analysis Workshop

Helen Lockstone, Bioinformatics Core

22 November 2019

Introduction

In this practical, we will work with RNA-Seq data for a subset of 50 samples from the Cancer Genome Atlas project (https://cancergenome.nih.gov). The 50 samples include 3 different cancer types (breast, kidney and endometrial) and we are interested to explore the differences between them using the R/Bioconductor package edgeR.

Setup

All material can be found at the course link:
https://www.well.ox.ac.uk/bioinformatics/training/RNASeq_221119

1. Create a new folder for today's workshop on your computer
2. Open a new R session by launching RStudio
3. Set the working directory in RStudio to match the location of the your new folder

This can be done via the RStudio main menu toolbar at the top: Session > Set working directory > Choose working directory…

Navigate through your filesystem to the correct directory and click OK. A command **setwd("path_to_directory")** should automatically be executed in the R console - the bottom left panel in RStudio. You can also confirm the correct setting of the working directory with the **getwd** command:

```
getwd()
```

R will by default look in the current working directory for files to read in, and the location to write any output files we generate.

You can run any commands shown in this document by copying the R code into the topleft panel of RStudio and clicking the icon with a green forward arrow in the toolbar. The line of code where the cursor is will be automatically copied into the bottom left panel and executed by R in realtime. You should see the R prompt ***>*** *again when it has finished. Depending on the command, some output may also be printed to the screen.*

If you see an error message, first double check that the command is entered exactly as shown. If you can't find the problem, please ask for help. The advantage of copying the commands into the text editor panel of RStudio means that you can save the file afterwards to have a record of what you have done and can easily re-run again in future.

Next, download the data files that we will be using by running the following two commands in R.

```
## file containing gene expression data
download.file("https://www.well.ox.ac.uk/bioinformatics/training/RNASeq_221119/data/Cancer_gene_expression_data.txt", "./Cancer_gene_expression_data.txt")

## file with sample information
download.file("https://www.well.ox.ac.uk/bioinformatics/training/RNASeq_221119/data/Cancer_sample_info.txt", "./Cancer_sample_info.txt")
```

The **"./name_of_file"** part tells R to save the file in the current working directory and the two files should now be visible there on your computer.

The final preparation step is to install and load a Bioconductor package for performing RNA-Seq analysis. We will use edgeR and you can read more details about this package here. See here for an overview of Bioconductor itself.

```
source("https://bioconductor.org/biocLite.R")
biocLite("edgeR")

library(edgeR)
```

Reading data into R and preparing for analysis

Now, we can read in the gene expression data file and check its contents - there should be 23368 rows corresponding to genes, and 50 columns corresponding to samples.

```
data <- read.table("Cancer_gene_expression_data.txt", sep="\t", header=T, row.names=1)
dim(data)

head(data[,1:6]) # shows the first 6 rows (default for head command) for the first 6 samples, which are selected using the 1:6 in square brackets
```

These are raw counts from an RNA-Seq experiment. Next read in the sample details from the second file.

```
s.info <- read.table("Cancer_sample_info.txt", sep="\t", header=T, stringsAsFactors=F)
dim(s.info)
head(s.info)
```

Note the argument stringsAsFactors=F in the command above to read in the sample information – without this, both columns would be treated as factors and behave differently in R. We could coerce to a vector (as.vector) but it's often safer to read in like this and create factors for columns you really want to treat as factors afterwards.

At this point, it is critical to ensure that the samples are in the same order in the data object and the accompanying sample information as future steps will assign experimental conditions assuming this to be the case.

To do this, we can compare the column names of our **data** object with the Sample_ID column of **s.info**. First, we need to deal with some automatic formatting R performed internally when reading in the data - if you look back at the output of the **head** commands, you will notice that the column names of **data** have dots replacing the original dash symbols in *Cancer_gene_expression_data.txt*.

Indeed the original file still contains the dashes, it is only altered in the R environment to ensure the column names conform to R's internal naming rules. Note this only applies to column names, and thus the dashes are also retained in column 1 of the ***s.info*** *object. Another rule regarding column names is that they must begin with a letter.*

To check if the two vectors match, we need to make the same dash to dot substitution in the sample IDs in the **s.info** object.

```
s.info$Sample_ID <- gsub("-", ".", s.info$Sample_ID)

## now check if the colnames of data and sample IDs in s.info match
identical(colnames(data), s.info$Sample_ID)
```

Unfortunately it returns FALSE, indicating a discrepancy somewhere and we need to check what it might be. (If you are quite familiar with R, have a think about how this might be done).

From a quick inspection of the first few samples, they look to be in the same order:

```
head(colnames(data))
head(s.info$Sample_ID)

## we can use the R function setdiff to help track down the issue:
setdiff(colnames(data), s.info$Sample_ID)
```

One of the original sample IDs, 3Z-A93Z, started with a number, which is not permitted for column names in R and so it has automatically been prefixed with an X to create a valid name. Therefore, we need to modify the sample table again for this specific sample:

```
match("3Z.A93Z", s.info[,1])
```

This tells us it is element (row) 32 of the first column of **s.info** that needs modifying. This can be done by assigning it the sample name as it appears in the column names of **data**.

```
s.info[32, 1] <- "X3Z.A93Z"

## finally re-running the previous command confirms they are now identical
identical(colnames(data), s.info$Sample_ID)
```

The formatting issues seen here are typical of many datasets where inconsistent naming patterns or special characters can cause problems for data handling and analysis. It cannot

be emphasised enough how important it is to check these details carefully before embarking on analysis steps, as it is very easy to make inadvertent mistakes that could render the resuslts meaningless.

Now everything is correct, we can proceed with setting up the analysis. Let's see what type of samples we have and how many of each.

```
table(s.info$Type)
```

BRCA indicates breast cancer samples, KIRC for kidney/renal and UCEC uterine/endometrial. Note that the total number of samples is 50, consistent with our R objects. Now we can define a factor for our experimental groups (cancer types).

```
conds <- factor(s.info$Type)

## inspect the new factor object
conds
```

Returning to the count data, we might also want to check the sequencing depth (number of millions of reads) for each sample.

```
read.depth <- apply(data, 2, sum)/1000000
summary(read.depth)
barplot(read.depth, las=2, cex.names=0.5)
```

The median sequencing depth is ~47 million reads, but there is high variability between samples as illustrated in the barplot.

*The **apply** function is a very efficient way to perform a particular task across columns of an object (indicated by the 2 in the command above) or rows (1), in this case computing the sum of all reads in each sample.*

Performing differential expression analysis with edgeR

The edgeR package was developed by Gordon K Smyth and colleagues at the Walter and Eliza Hall Institute in Melbourne around 10 years ago and has been extended and improved ever since. The same group previously developed the widely-used 'limma' framework for microarray data and many of the concepts were applied to RNA-Seq data. edgeR implements statistical models suitable for count data, namely the negative binomial model. It performs the computations very efficiently and can handle complex experimental designs through a generalized linear model (glm) framework. It also performs a normalisation step that accounts for the behaviour of RNA-sequencing in specific situations, such as when a subset of genes are highly expressed in one of the experimental conditions but not another; this can lead to over-sampling of the highly-expressed genes in one condition leaving other genes relatively under-sampled. If this is not corrected for in the normalisation step, many

of those other genes would appear artificially down-regulated in the condition with the highly-expressed genes.

Loading the edgeR package in the current R session as we did at the start of the workshop gives us access to all the required functionality. The authors of R/Bioconductor packages wrap the detailed steps of their methods into individual functions, which are usually private to a specific package, and will be available to R once the package has been loaded into the current session. From the user's perspective, the analysis is then performed by running a series of functions (steps) tailored to your particular dataset. All packages have documentation for their usage, and the edgeR package is particuarly well-documented with a detailed User Guide, which includes a number of case-study examples.

As with all R packages, care must be taken to run all the steps in the right order and ensure any parts requiring manual set up are correct. In the context of gene expression analysis, a particularly important part to get right is the design matrix and ensuring the coefficient or contrast for the comparison(s) of interest are correctly extracted afterwards.

Pre-processing steps

Creating an edgeR data object

```
## read data into edgeR object
y <- DGEList(counts=data, genes=row.names(data))
head(y$counts[,1:6]) # inspect the first 6 samples

## the counts per million (cpm) values have been computed and stored as well
head(cpm(y)[,1:6])
## they can be saved to a file if desired
write.table(cpm(y), "Cancer_dataset_counts_per_million.txt", sep="\t",
quote=F, row.names=T)
```

Our data is now stored in a DGEList object - this object class has been defined in the edgeR package to handle specific features of RNA-Seq data, and store the output of functions performed in the analysis process. It is a list object, which at this stage has 3 elements. These have pre-defined names and each component can be accessed using the $ symbol after the name of the DGEList object, **y**.

```
names(y)
head(y$counts)
```

Use the **head** command to inspect the other components of the object. What do you think they represent?

Filtering out low expressed genes

A common step in gene expression data analysis is to remove genes with zero or very low read counts. In a typical RNA-Seq dataset, many genes have zero counts in all samples and these cannot be analysed at all. They represent genes either unexpressed in the particular cell type and condition under study, or at such low levels such that they are not sampled in

the sequencing process. If there are only a handful of reads for a given gene, it is also difficult to make any statistical inference about differential expression and so a threshold of >10 reads might be applied. For more explanation of the filtering step, please click here

Filtering can be performed as follows:

```
## defining a filter to remove low expressed genes
## median depth is ~50m reads per sample
## raw count >10 corresponds to 0.2 cpm (counts per million) in this dataset
## smallest group size is 9
## retain genes with cpm>=0.2 in at least 9 samples
## note filter is independent of sample group (can be any 9 samples)


keep <- rowSums(cpm(y)> 0.2) >= 9
```

This line of code is rather clever and very efficient. It is used in the edgeR case studies* but it is not intuitive as to what it is doing. Usually the **rowSums** function will compute the sum of each row in a matrix. The inclusion of the >0.2 returns instead the number of elements (i.e. samples) in that row for which the condition is met. The final >=9 further changes the nature of the output returned by the function, and this time returns TRUE if the condition is met in 9 or more samples and FALSE otherwise.*

*At least it used to be, there is now an in-built function called **filterByExpr**. You can explore how this works with:

```
help("filterByExpr")
```

You can see this behaviour by building up the command in stages:

```
head(cpm(y)[,1:6]) # shows the CPM values for part of the dataset

head(rowSums(cpm(y))) # shows the sum of CPM values across each row for the
first few rows

head(rowSums(cpm(y) > 0.2)) # shows the number of samples with values above
0.2 in each row

head(rowSums(cpm(y) > 0.2)) >= 9 # TRUE/FALSE vector for whether more than 9
samples in each row have CPM>0.2 or not
```

We can determine how many genes pass the filter using the ever-useful table function:

```
table(keep)
```

So 18483 genes passed the filter, while 4885 did not. Relatively few genes are excluded with this filter, which may be linked to the high sequencing depth or the source of gene annotations (RefSeq vs Ensembl).

We now need to extract only the genes passing the filter from our data object

```
y <- y[keep,]
dim(y)
```

Data normalisation

We can now proceed to normalising the data using edgeR's bespoke method, TMM, the **trimmed mean of M-values**. (M-values are otherwise known as the fold-changes, with the terminology originally used for microarray data).

```
## calculate normalisation factors
y <- calcNormFactors(y)
class(y)

names(y)

head(y$counts[,1:6]) # header of the count matrix for the first 6 samples
head(y$samples) # sample details including library size i.e. total reads or
depth and the normalisation factors
```

Notice how the normalisation factors in the object have been updated from their default values of 1 before the normalisation step was run.

Data exploration

edgeR also provides a visualisation of sample clustering with a multi-dimensional scaling plot.

```
plotMDS(y)

## refining the plot to label by cancer types rather than sample IDs

plotMDS(y, labels=conds, cex=0.6)
```

This plot shows strong clustering according to the cancer type, which suggests we will find many signficantly differentially expressed genes in the analysis step.

Bear in mind that this exploratory plot is based on the overall gene expression profile and even if strong clustering is not observed for a particular dataset, there may well still be signifcant genes to be found in the analysis.

Testing for differential expression

Defining the analysis model

For this step, it is important to be familiar with R's methods for defining linear models and determining what the coefficients represent.

First we need to create a design matrix based on the experimental design, which can usually be set up in several equivalent ways. Here we will define the design matrix such that each column corresponds to one of the cancer types. Each sample has a row in the design matrix, and the columns indicate which samples belong to each experimental group: 1 when the sample belongs to that group, 0 otherwise.

This means our model will have 3 coefficients, each estimating the mean expression in one of the groups. We can then use contrasts to find the difference between pairs of coefficients to find differentially expressed genes between cancer types. This is usually the safest way to define the model, especially for more complex experimental designs. A full tutorial on this topic from last year's course is available here, adapted from relevant sections of the limma Users Guide Chapter 9. There are also more details and examples in the edgeR User Guide.

```
design <- model.matrix(~0+conds)
head(design)
colnames(design) <- gsub("conds", "", colnames(design))
head(design)
```

Can you see why the columns have been named in the order they appear in the design matrix? *Hint, inspect the* ***conds*** *object again.*

Recall from earlier that our factor **conds** describing the status of each sample was obtained from the information read in from the sample information file and stored in **s.info**. The design object that we have just created uses **conds** to define the cancer type of each sample in the model. Thus, the first 3 samples are all breast cancer, the 4th is uterine, the 5th is breast etc. We can double check that this is consistent with our sample details.

```
head(s.info)
```

This is also why the check for the order of samples in the **data** object matching that of **s.info** was so important: the rows of the design matrix are assumed by R to correspond to the order of columns in the **data** object that we later use to fit the model.

Estimating gene dispersions

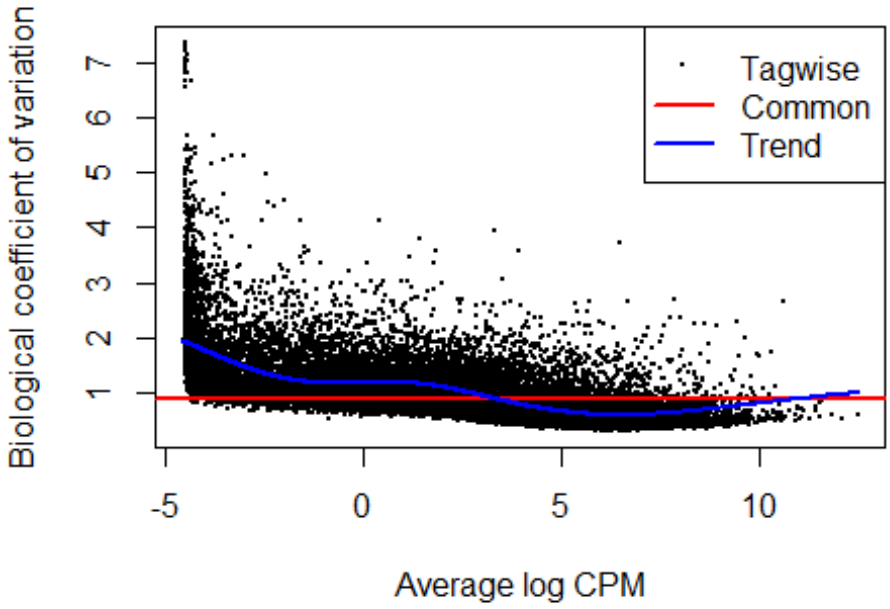
The variance of a gene across samples is a crucial factor in determining how significant a change in gene expression is - see Exercise 1 of this tutorial for a demonstration of this. edgeR provides a common dispersion estimate for all genes, a trended estimate by mean expression, and individually for each gene. By default, edgeR will make use of the gene-specific estimates, which are now thought to be the most appropriate choice for assessing differential expression. They are also moderated to make them more robust to the typically small sample size of expression studies (see Exercise 2).

```
y <- estimateGLMCommonDisp(y, design, verbose=T)

## Disp = 0.77423 , BCV = 0.8799

y <- estimateGLMTrendedDisp(y, design)
y <- estimateGLMTagwiseDisp(y, design)
```

```
plotBCV(y)
```



Fitting the model

This step fits the negative binomial (NB) glm to each gene separately. Even for thousands of genes, it only takes a few moments to run.

```
fit <- glmFit(y, design)
```

As mentioned before, the way we have set up the model does not estimate the differences between groups directly, but each coefficient estimates the mean expression level in one of the three cancer groups.

We can define a contrast matrix to perform the comparisons of interest - pair-wise comparisons of the 3 groups. Note we can refer to the groups by the column names of the design matrix.

```
cont.matrix <- makeContrasts(BRCA-KIRC, BRCA-UCEC, KIRC-UCEC, levels=design)
cont.matrix
```

The first contrast is BRCA vs KIRC, indicated by the 1 and -1 in the first column of the matrix.

Extracting the BRCA vs KIRC comparison

```
res.brca.kirc <- glmLRT(fit, contrast=cont.matrix[,1])
```

```
class(res.brca.kirc)
names(res.brca.kirc)
```

This step completes the testing for differential expression and stores the results in a new object we named **res.brca.kirc**. This object contains a large number of components relating to the statistical testing. The main results table can be accessed with:

```
head(res.brca.kirc$table)
```

This is a similar format to the topTable output of limma, and contains the results for all analysed genes. Notice that they are in alphabetical order at this point and also that only raw p-values are given. The testing procedure generates a likelihood ratio (LR) as a test statistic and corresponding p-values.

To rank the genes according to evidence for differential expression between the BRCA and KIRC samples we can run the following commands.

```
o <- order(res.brca.kirc$table$PValue)
res.brca.kirc <- res.brca.kirc$table[o,]
```

It is common practice to adjust the raw p-values for multiple testing (of thousands of genes) with a false discovery rate correction. More explanation on p5 of this tutorial

```
## compute and add adjusted p-values
adjp <- p.adjust(res.brca.kirc$PValue, method="fdr")
res.brca.kirc <- cbind(res.brca.kirc, adjp)
head(res.brca.kirc)
write.table(res.brca.kirc, "Cancer_dataset_BRCA_vs_KIRC_edgeR_results.txt",
sep="\t", quote=F, row.names=T)
```

This saves the entire output to file - it is always worth writing out the full table of genes as it can be filtered to significant genes later, but if you want to check the result for a particular gene (maybe it just missed the significance threshold) and had only saved the significant genes, you would need to run all the analysis code again.

Notice that the file naming convention is consistent with the way our contrasts were defined earlier and helps interpret the logFCs correctly - they could be cross-checked against raw data but otherwise there would be no way of knowing which group was used as the reference. Again by convention, the logFCs are log2 scale with 0 indicating no difference between groups, positive logFCs indicate higher expression in the first group, and negative logFcs indicate higher expression in the second group.

We can also summarise the genes called significantly differentially expressed.

```
de <- decideTestsDGE(res.brca.kirc)

## by default summarises results at 5% FDR

summary(de <- decideTestsDGE(res.brca.kirc))
```

This shows the split of up and down-regulated genes, and should match the total number of genes with adjusted p-values <0.05:

```
length(which(res.brca.kirc$adjp<0.05)) # 7434
```

Finally, we can visualise the results for each gene by plotting mean expression level vs logFC.

```
detags <- rownames(y)[as.logical(de)]
plotSmear(res.brca.kirc, de.tags=detags, main="BRCA vs KIRC", ylim=c(-10,10))
abline(h=c(-1,1), col="blue")
```

Exercise

Repeat the last section to extract differential expression results for:

1. Breast vs uterine cancer samples
2. Kidney vs uterine cancer samples

Store each set of results in a suitably-named object and note how many significant genes there are in each list. Use the **intersect** function to determine the overlap between each pair of lists.

Data visualisation

There are various ways that the results can be visualised and further explored for biological insights.

For example, we can make boxplots for the top 10 differentially expressed genes for the breast vs kidney comparison and save them to a PDF. The output file can be opened from your working directory once this segment of code has been run.

```
cpm <- as.matrix(cpm(y))
pdf("Cancer_dataset_boxplots_top10_genes_BRCAvsKIRC.pdf", onefile=T)
# use a loop to iterate over 10 genes
for(i in 1:10)
{
    x <- match(row.names(brca.kirc)[i], row.names(cpm))
    boxplot(cpm[x,]~conds, las=2, cex.axis=0.8,
main=paste0(row.names(brca.kirc)[i]), ylab="Counts per million")
}
dev.off()
```

We can also make a heatmap for the top 100 genes that were differentially expressed in the breast vs kidney comparison. Log-scaled cpm values are reasonable input to the heatmap. the main expression matrix (uses ‘match’ to identify which rows to keep)

```
d.plot <- cpm[match(row.names(res.brca.kirc)[1:100], row.names(cpm)),]
dim(d.plot) ## 100   50
class(d.plot) ## matrix

d.plot.log <- log2(d.plot+1)

## this cpm matrix can now be used to make a heatmap of the expression levels
of these genes in all 50 samples

## we need another package for this, such as 'gplots'

library(gplots)
heatmap.2(d.plot, col=greenred(75), scale="row", dendrogram="both",
density.info="none", trace="none", keysize=0.5, labCol=conds, lmat=rbind(
c(0, 3), c(2,1), c(4,4) ), lhei=c(1.5,5.5,1.5), main="Heatmap of top 100
genes, BRCA vs KIRC", cexCol=0.8, cexRow=0.7)

## save the plot to file
pdf("Cancer_dataset_heatmap.pdf")
heatmap.2(d.plot.log, col=greenred(75), scale="row", dendrogram="both",
density.info="none", trace="none", keysize=0.5, labCol=conds, lmat=rbind(
c(0, 3), c(2,1), c(4,4) ), lhei=c(1.5,5.5,1.5), main="Heatmap of top 100
genes, BRCA vs KIRC", cexCol=0.8, cexRow=0.5)
dev.off()
```

The heatmap code needs refining to make a more aesthetic figure. See if you can make any improvements to it.

Pathway analysis

edgeR offers some ways to explore the functional roles of genes found differentially expressed, as do many other R/Bioconductor packages such as Goseq or tools such as GSEA from the Broad Institute, which is also implemented in R. The biomaRt package is very useful for mapping between and retrieving different gene annotations and identifiers.

Further reading

The original limma package for microarray data was subsequently extended to include methods to deal with RNA-Seq data by transforming the count data prior to fitting standard linear models (the alternative to using models specifically for count data). There was some dilemma over which approach would be best and in the end they did both! This is the limma-voom approach and is described in Chapter 15 of the limma Users Guide. If you are really keen, you can run the same analysis with limma-voom and see how the results compare.

There are also a host of other packages and methods for analysing RNA-Seq data - we have focused on edgeR as it is the one we primarily use, largely based on the strong reputation and track record of its developers.