RNA-Seq Data Analysis Course

Practical: Gene expression analysis in R

Tuesday 27th November 2018

All material can be found at the course link

http://www.well.ox.ac.uk/bioinformatics/training/RNASeq_Nov2018/Day2_271118

First, download the data files 'Cancer_gene_expression_dataset.txt' and 'Cancer_sample_info.txt' from the subfolder *Files_and_scripts* to your computer, and make a note of the directory where you put them.

Open a new R session, and set that directory as your 'working' directory (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 in this practical). If you have any difficulties saving the files or setting the working directory, please just ask for help. Commands to enter or copy and paste into your R session are shown in *this font and colour.*

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 packages limma and edgeR. They can be installed by typing the following in your R session:

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

Once installed, load the libraries into the current session with:

```
library(limma)
library(edgeR)
```

Note the installation of a particular package only needs doing once (unless it is updated or when you upgrade to a newer version of R). You can subsequently just run the library command to load it into any particular R session

Alternatively, you can use the 'require' command checks if the package is already installed (and loads it if so) and automatically downloads it if not.

```
require(limma)
require(edgeR)
```

First, download the file 'Cancer_gene_expression_dataset.txt' to your preferred working directory and ensure this is the current working directory in R. Read in the 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_dataset.txt", sep="\t", header=T, row.names=1)

dim(data) # 23368   50

head(data)
```

these are raw counts from an RNA-Seq experiment

sample details are in a separate file 'Cancer_sample.info.txt', also download this file to your working directory

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

dim(s.info) # 50  2

head(s.info)
```

Check sample information file is in the same order as data matrix – note that when R read in the expression data file, the dash symbols (-) in the original column names (sample IDs) were automatically replaced with dots (.) to conform to R's internal naming rules.

To check if the two vectors match, we need to make the same substitution in the sample IDs in the s.info object. Note the argument stringsAsFactors=F in the command above to read in the sample information – without this, the first column would be read in as a factor and behave differently. We could coerce it 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.

The 'identical' command checks whether the column names of our data object *exactly* match the sample IDs listed in the sample file, after substituting dashes with dots:

```
identical(colnames(data), gsub("-", ".", s.info$Sample_ID)) # FALSE
```

Unfortunately it returns FALSE, indicating a discrepancy somewhere and we need to check what it might be:

```
head(colnames(data))

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

These look to be in the same order. Let's use setdiff to help track down the issue:

```
setdiff(colnames(data), as.vector(gsub("-", ".", s.info$Sample_ID)))
```

[1] "X3Z.A93Z"

One of the sample IDs started with a number, which is not permitted for column names in R - it has automatically been prefixed with an X to create a valid name. Therefore, we need to modify the sample info table to match the column names of 'data':

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

[1] 32

This tells us it is element (row) 32 of the first column of s.info that needs replacing

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

Re-running the test for matching names:

identical(colnames(data), as.vector(gsub("-", ".", s.info$Sample_ID))) # TRUE

Now everything is correct, we can proceed with setting up the analysis.

get a breakdown of cancer types

table(s.info$Type)

BRCA KIRC UCEC

29 9 12

BRCA indicates breast, KIRC kidney/renal and UCEC uterine/endometrial cancer types

define a factor describing the cancer types (groups to be compared later)

conds <- factor(s.info$Type)

inspect the created factor object

conds

returning to the count data, 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)

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

We can use the 'voom' function in limma to transform the count data so that is suitable for input in the linear model framework (see chapter 15 of limma usersguide). This assumes that zero and low count genes have been removed from the dataset, so let's do that now.

Filtering to remove low-expressed genes

A good rule of thumb is to consider genes as low expressed when there are typically fewer than 10 reads per sample for that gene. Note that this type of filter is designed to remove genes with low expression across ALL samples, while retaining those that might be highly expressed in one experimental group but unexpressed in another. It is also important to clarify that any filtering is performed blind to the experimental group classes, to avoid any bias. Thus, the structure of the filter is usually something like 'keep genes with counts > 10 in at least n samples' where n corresponds to the size of the smallest group e.g. if an experiment consisted of 4 groups, each with n=3 replicates, we might want to retain genes expressed in one or more groups, while excluding those that were low in all 4 groups (because these are not likely to be differentially expressed or of interest, and indeed probably have too few reads to perform any reasonable inference on in any case). Our filter would count how many samples had >10 reads for a given gene, and if that were more than 3 samples, keep the gene for further analysis. Note these can be any 3 samples (to keep the filtering unbiased) but among them would include genes whose expression was high in the 3 replicates of a given condition. Similarly, genes expressed in 2 of the 4 groups would also pass the filter and so on.

Particularly when read depth is quite variable between samples, it is preferable to perform the filtering on counts per million (cpm) data and the edgeR package (https://bioconductor.org/packages/release/bioc/html/edgeR.html) provides an easy way to do this, as well as further normalisation between samples (TMM - trimmed mean of M-values (fold changes), see the edgeR users guide for more details).

To calculate the filter threshold, we need to determine the cpm value that equates to ~10 reads. If there were typically 10 million reads per sample, cpm=1 would be the equivalent of 10 raw reads, and if there were 25 million reads per samples, cpm=0.4 corresponds to 10 raw reads, so the cpm threshold changes depending on the sequencing depth of the experiment. The conversion is: raw read threshold/average depth (in millions) so for the second example above: 10/25 = 0.4 cpm.

In our cancer dataset, we'll round up to 50m as our typical read depth: 10/50 = 0.2 cpm for our filter.

```
## read data into edgeR object

y <- DGEList(counts=data, genes=row.names(data))

head(y$counts)

head(cpm(y))

write.table(cpm(y), "Cancer_dataset_counts_per_million.txt", sep="\t", quote=F, row.names=T)


## filter to remove low expressed genes

## median is ~50m reads per sample

## raw count >10 corresponds to 0.2 cpm (counts per million)

## smallest group size is 9
```

note filter is independent of sample group information (can be any 9 samples)

the following line of code returns a logical vector (TRUE/FALSE) for our filter condition: is the cpm value > 0.2 in 9 or more samples (TRUE if this is the case).

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

table(keep)

FALSE TRUE

4885 18483

The 'table' function shows that 18483 genes passed the filter, while 4885 did not. We can retain those as follows:

y <- y[keep,]

dim(y) # 18483   50

calculate normalisation factors (trimmed mean of M-values, TMM, method implemented in edgeR)

y <- calcNormFactors(y)

class(y)

[1] "DGEList"

attr(,"package")

[1] "edgeR"

Our data object is a special kind of structure of class DGEList - this class has been defined in the edgeR package to handle specific features of RNA-Seq data, and store the output of functions calculated in a typical analysis process (such as library sizes (depth) and normalisation factors). It is a list object with (currently) 3 elements - we can access them by name, and can find out the names with

names(y)

[1] "counts" "samples" "genes"

head(y$counts) # the count matrix

head(y$samples) # sample details including library size (i.e. total reads or depth) and the normalisation factors

define experimental design using the group-means parameterization i.e. each column of the design matrix estimates mean expression in one of the cancer types

design <- model.matrix(~0+conds)

head(design)

colnames(design) <- gsub("conds", "", colnames(design))

head(design)

Performing analysis with limma voom

use limma 'voom' function to transform the data to suitable input for usual limma models

max(read.depth)/min(read.depth) # 7.25

confirms that 'voom' is the best option as the ratio of the max/min read depths is >3 (see chapter 15 of limma users guide for details)

v <- voom(y, design, plot=TRUE)

if we check what type of object the 'voom' function produces, we see another package-specific list structure, this time called EList

class(v)

[1] "EList"

attr(,"package")

[1] "limma"

names(v)

head(v$E) # transformed expression data

Having suitably transformed the data, we can proceed to use limma functions to analyse the data:

perform differential expression analysis, using contrasts to extract the pairwise comparisons of the 3 types

fit.voom <- lmFit(v, design)

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

cont.matrix

fit2.voom <- contrasts.fit(fit.voom, cont.matrix)

fit2.voom <- eBayes(fit2.voom)

use topTable to output the results for BRCA vs KIRC and store in a data frame named according to the comparison performed (note the coef argument takes numbers corresponding to the columns of cont.matrix)

```
brca.kirc <- topTable(fit2.voom, coef=1, number=nrow(y$counts), sort.by="p", adjust.method="BH")

length(which(brca.kirc$adj.P.Val < 0.05)) # 6458
```

Save the entire output to file – it's always worth writing out the full table of genes as it can always 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 only saved the significant genes, you'd need to run all the analysis code again.

notice that the file naming convention is consistent with the way our contrasts were defined earlier.

```
write.table(brca.kirc, "Breast_vs_Kidney_limma_output.txt ", sep= "\t", quote=F, row.names=T)
```

make boxplots for the top10 differentially expressed genes and save them as a PDF. The file will be created in your current working directory (use getwd() to check what this is), and can be opened for inspection once this segment of code has been run.

```
cpm <- as.matrix(cpm(y))

 pdf("Cancer_dataset_boxplots_top10_genes_BRCAvsKIRC.pdf", onefile=T)

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()
```

Analysis using edgeR for differential expression

As an alternative to limma voom, the analysis could be performed entirely in edgeR using count-based statistical models for testing for differential expression (https://www.bioconductor.org/packages/release/bioc/html/edgeR.html). Let's do this as well and compare the results. We can make use of the same design matrix and contrasts matrix we created earlier, and use our edgeR DGEList object, y, for the input to the next steps:

estimate overall dispersion

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

Disp = 0.67548 , BCV = 0.8219

fairly high dispersion values, presumably as human cancer samples

estimate gene-wise dispersion

y <- estimateGLMTrendedDisp(y, design)

y <- estimateGLMTagwiseDisp(y, design)

plotBCV(y)

testing for diff expression

fit <- glmFit(y, design)

implements a likelihood ratio test to determine significance - can use the cont.matrix created earlier to define comparisons of interest:

cont.matrix

	Contrasts		
Levels	brca_kirc	brca_ucec	kirc_ucec
BRCA	1	1	0
KIRC	-1	0	1
UCEC	0	-1	-1

Extracting the BRCA vs KIRC comparison again:

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

this step completes the testing for differential expression and stores the results in res.brca.kirc.

class(res.brca.kirc)

[1] "DGELRT"

attr(,"package")

[1] "edgeR"

> names(res.brca.kirc)

[1] "coefficients" "fitted.values" "deviance"

[4] "method" "unshrunk.coefficients" "df.residual"

[7] "design" "offset" "dispersion"

[10] "prior.count" "samples" "genes"

[13] "prior.df" "AveLogCPM" "table"

[16] "comparison" "df.test"

This contains a large number of components relating to the statistical analysis. The results table can be accessed with:

head(res.brca.kirc$table)

	logFC	logCPM	LR	PValue
X1.2.SBSRNA4	0.03016899	-0.2307173	7.181025e-03	9.324674e-01
A1BG	0.80834173	2.7301765	2.498438e+00	1.139593e-01
A1BG.AS1	0.95355021	0.3968704	3.956133e+00	4.670086e-02
A1CF	-8.06573159	2.8617632	1.501434e+02	1.612899e-34
A2LD1	-0.18102358	2.5671025	2.907771e-01	5.897230e-01
A2M	-0.18292652	10.3411577	1.352376e-01	7.130621e-01

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. edgeR implements Generalized linear models to handle the count data generated by RNA-Seq; 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:

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

out <- res.brca.kirc$table[o,]
```

add adjusted p-values

```
adjp <- p.adjust(out$PValue, method="fdr")

out <- cbind(out, adjp)

head(out)

write.table(out, "Cancer_dataset_BRCA_vs_KIRC_edgeR_results.txt",      sep="\t", quote=F, row.names=T)
```

We can also summarise the genes called significantly differentially expressed:

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

results at 5% fdr

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

1*BRCA -1*KIRC	
-1	3313
0	11049
1	4121

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(out$adjp<0.05)) # 7434
```

Finally, we can visualise the results on a plot of 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")
```

Overlap between limma voom and edgeR results

check the overlap of the two gene lists for BRCA vs KIRC

sig.voom <- row.names(brca.kirc)[1:6458]

head(sig.voom)

[1] "C14orf105" "SLC22A2" "BHMT" "TINAG" "NR1H4"

[6] "LOC100422737"

sig.edgeR <- row.names(out)[1:7434]

head(sig.edgeR)

[1] "SLC22A2" "BHMT" "ENPEP" "SLC17A3" "SLC17A1" "NDUFA4L2"

length(intersect(sig.voom, sig.edgeR))

[1] 5724

Optional exercise

What do you conclude about the two approaches (limma-voom and edgeR)? What reasons can you think of for genes being identified as significant with one approach but not the other?

Write some R code to identify where the significant genes from one approach are ranked in the list of results for the other approach.

Visualising data using a heatmap

We might also like to make a heatmap for the 100 most variable genes in the dataset (likely to be differentially expressed between at least one pair of cancer types)

we can use the 'apply' function to calculate the variance of each row (gene) and store in a vector named 'data.var'

data.var <- apply(v$E, 1, var)

sort from most to least variable

data.var <- sort(data.var, decreasing=TRUE)

head(data.var)

SCGB2A2 ANKRD30A MUCL1 CDH16 FABP7 SLC17A3

36.28524 30.82677 28.24454 27.45653 25.44809 25.42329

var.genes <- names(data.var)[1:100]

length(intersect(var.genes, row.names(v$E))) # 100 - checking all the names are found in the data object

now extract this subset of 100 genes from the main expression matrix (uses 'match' to identify which rows to keep)

```
d.plot <- v$E[match(var.genes, row.names(v$E)),]

dim(d.plot) ## 100   50

class(d.plot) ## matrix
```

this data 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' (available from CRAN with the following command:

```
install.packages("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 100 most variable genes", cexCol=0.8, cexRow=0.7)
```

save the plot to file

```
pdf("Cancer_dataset_heatmap.pdf")

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 100 most variable genes", cexCol=0.8, cexRow=0.5)

dev.off()
```