RNAseq differential expression analysis using the DESeq package

Slides by: Dr Jelena Aleksic (ja313@cam.ac.uk)

Practical schedule

1. Loading the DESeq library
2. Data loading
3. DESeq count object and calculations
4. Results filtering
5. Writing out and exploring results

Loading the DESeq library

1

What is DESeq?

•From an RNAseq experiment, we get counts data showing how many counts are mapping to each gene.

•This data follows a standard distribution (negative binomial), and can be statistically modeled.

•This way we can determine which genes are differentially expressed between different conditions in a statistically significant manner.

•There are a number of different packages, in R and otherwise, that deal with this problem. A few good examples:

- DESeq
- EdgeR

This practical focuses on performing a differential expression analysis using DESeq.

DESeq

Bioconductor
OPEN SOURCE SOFTWARE FOR BIOINFORMATICS

Differential gene expression analysis based on the negative binomial distribution

Bioconductor version: Release (3.0)

Estimate variance-mean dependence in count data from high-throughput sequencing assays and test for differential expression based on a model using the negative binomial distribution

Author: Simon Anders, EMBL Heidelberg <sanders at fs.tum.de>

Maintainer: Simon Anders <sanders at fs.tum.de>

Citation (from within R, enter `citation("DESeq")`):

> Anders S and Huber W (2010). "Differential expression analysis for sequence count data." *Genome Biology*, **11**, pp. R106. http://dx.doi.org/10.1186/gb-2010-11-10-r106, http://genomebiology.com/2010/11/10/R106/.

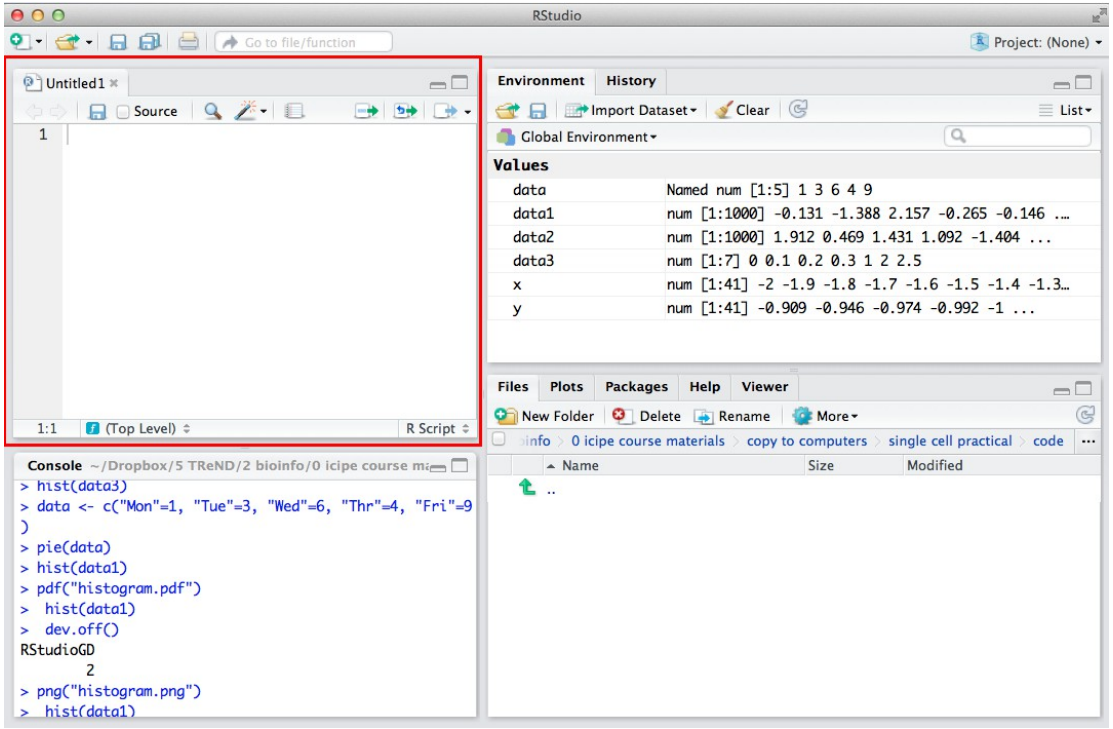
Installation

To install this package, start R and enter:

```
source("http://bioconductor.org/biocLite.R")
biocLite("DESeq")
```

Writing a DESeq script

- This analysis is long, and will get very confusing if we write it out entirely on the console.
- The script I use day to day is 280 lines long. Imagine making an error in line 50, and having to retype everything again...
- Instead, write a script that you can rerun!



Loading the DESeq library

- At the very beginning of your script, it is usual to load all the libraries that you will be using in your analysis
- To load DESeq, type:

```
library("DESeq")
```

The library is now loaded, which means that you can use new custom functions included in it.

Data loading

2

3 steps to
Basic data analysis

1. Reading in data
    - read.table()
    - read.csv(), read.delim()

2. Analysis
    - Manipulating & reshaping the data
    - Any maths you like
    - Plotting the outcome
        - High level plotting functions (covered tomorrow)

3. Writing out results
    - write.table()
    - write.csv()

Step 1.
Read in the data

Transcript ID	mother_c_tr	WT_a_transcript	mother_d_trar	WT_b_trans	WT_c_transc
ENSG00000000003	144	138	146	148	176
ENSG00000000005	0	0	0	0	0
ENSG00000000419	224	203	213	223	194
ENSG00000000457	107	109	87	123	146
ENSG00000000460	51	53	35	52	53
ENSG00000000938	0	0	0	0	2

This data is a tab delimited text file.
Each row is a gene, each column gives the read counts for that gene in a given replicate.

We need to read in the results table and assign it to an object (rawData)

```
rawData <- read.table("counts_table.txt", header=T, row.names=1, sep="\t")
head(rawData) # View the first few rows to ensure import is OK
              # Check that the header and rownames are correct
```

If the data had been comma separated values, then sep=","

```
read.csv("counts_table.csv")
```

`?read.table` for a full list of arguments

DESeq_analysis.R
(script commands)

counts_table.txt
(data file)

Renaming data columns

- We want to be able to navigate the data more easily. The column names are long at the moment.
- We will reassign the names:

```
> names(rawData) <- c("WT_a", "WT_b","WT_c","WT_d","patient_a",
                      "patient_b","patient_c","patient_d")
```

- We then check the first few rows again to make sure data looks ok.

```
> head(rawData)
```

	WT_a	WT_b	WT_c	WT_d	patient_a	patient_b	patient_c	patient_d
ENSG00000000003	138	148	176	164	122	117	106	109
ENSG00000000005	0	0	0	0	0	0	0	1
ENSG00000000419	203	223	194	225	200	180	209	181
ENSG00000000457	109	123	146	133	95	111	110	108
ENSG00000000460	53	52	53	69	43	48	46	33
ENSG00000000938	0	0	2	2	0	0	0	0

DESeq counts object and calculations

3

Reading data into a DESeq counts object

- In programming, "objects" lump together collections of data about a specific data type.
- DESeq has a special object category specifically for counts data
- Minimally, it contains two things:
    - The counts
    - The experiment conditions that the counts came from
- First we put in the conditions:

```
> conds <- c(rep("WT",4), rep("patient",4))
```

- We then create the data object using the raw data+conditions

```
> cds <- newCountDataSet(rawData, conds)
```

- From here, we can run special DESeq functions on the counts data object.

Scaling factor normalization

- If the different replicates have different sizes of libraries, this is something we need to normalize for
- On the simplest level, if one library was 10M, and the other one was 20M, you would divide by **2** to normalize.
- However, if there are genes that are highly expressed under one condition, but not the other, this can skew the normalization.
- DESeq finds the genes that do not change expression under different conditions, then uses these to calculate scaling factors.
- There is a quick command for doing this:

```
> cds <- estimateSizeFactors( cds )
```

- You can view the scaling factors once they've been calculated:

```
> sizeFactors( cds )
     WT_a      WT_b      WT_c      WT_d patient_a patient_b patient_c
patient_d
0.9703356 1.0301293 1.1092747 1.0184672 1.0266645 0.9440706 1.0554744
0.9068650
```

Saving normalized counts

- When we do the normalization, it is useful to save the normalized counts in a variable, because these are the counts we would use for plotting QC graphs.

```
> nCounts <- counts(cds, normalized=TRUE)
```

- We can also write these out in a text file, so that we can load them in for future use:

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

- This gives us a tab delimited table of results. We have it saved for future reference, and can explore it in a spreadsheet, etc.

Estimating variance and statistical testing

- In order to run the tests, DESeq first estimates dispersions:

```
> cds = estimateDispersions( cds )
```

- You can look up information about the model fit:

```
> str( fitInfo(cds) )
List of 5
 $ perGeneDispEsts: num [1:62757] -0.001664 0.774016 0.00065 0.000405
0.003312 ...
 $ dispFunc       :function (q)
  ..- attr(*, "coefficients")= Named num [1:2] 0.00276 0.54945
  .. ..- attr(*, "names")= chr [1:2] "asymptDisp" "extraPois"
  ..- attr(*, "fitType")= chr "parametric"
 $ fittedDispEsts : num [1:62757] 0.00687 3.98898 0.0055 0.0075 0.01394 ...
 $ df             : int 6
 $ sharingMode    : chr "maximum"
```

- You can also plot the dispersions, which shows you the distribution of the data and the fit of your model:

```
> plotDispEsts( cds )
```

Assessing differential expression

- After this, we are ready to assess the differential expression of different genes. DESeq does this by fitting a negative binomial distribution to each gene.

```
> res = nbinomTest( cds, "WT", "patient" )
```

- We can have a look at the results table:

```
> head(res)
```

```
                id    baseMean   baseMeanA  baseMeanB foldChange
log2FoldChange        pval         padj
1 ENSG00000000003 133.6205697 151.3946652 115.846474  0.7651952
-0.38610023 0.001441248 0.007606614
2 ENSG00000000005   0.1378375   0.0000000   0.275675        Inf
Inf 0.984187482 1.000000000
3 ENSG00000000419 200.5707707 205.3732346 195.768307  0.9532318
-0.06910095 0.549740246 0.952033382
4 ENSG00000000457 115.9199308 123.4851744 108.354687  0.8774712
-0.18857628 0.130015773 0.354195961
5 ENSG00000000460  49.1656843  55.1568046  43.174564  0.7827604
-0.35335727 0.072773617 0.224791201
6 ENSG00000000938   0.4708394   0.9416788   0.000000  0.0000000
-Inf 0.240206591 0.5572368023
```

Results filtering

4

Filtering the results table

- There are some values in the table that are useless to us – for example, the "Inf" fold change caused because one of the conditions had zero reads. To clean up the table first we want to filter away all the incomplete / useless results.

```
> head(res)
```

```
                id    baseMean   baseMeanA  baseMeanB foldChange
log2FoldChange        pval         padj
1 ENSG00000000003 133.6205697 151.3946652 115.846474  0.7651952
-0.38610023 0.001441248 0.007606614
2 ENSG00000000005   0.1378375   0.0000000   0.275675        Inf
Inf 0.984187482 1.000000000
3 ENSG00000000419 200.5707707 205.3732346 195.768307  0.9532318
-0.06910095 0.549740246 0.952033382
4 ENSG00000000457 115.9199308 123.4851744 108.354687  0.8774712
-0.18857628 0.130015773 0.354195961
5 ENSG00000000460  49.1656843  55.1568046  43.174564  0.7827604
-0.35335727 0.072773617 0.224791201
6 ENSG00000000938   0.4708394   0.9416788   0.000000  0.0000000
-Inf 0.240206591 0.5572368023
```

Thresholds and filtering incomplete cases

- There are some values in the table that are useless to us – for example, the "Inf" fold change caused because one of the conditions had zero reads. To clean up the table, we want to filter away all the incomplete / useless results.
- We also need to filter by p-value to get the significant results.
- Optionally, we can also filter all genes with less reads than a particular threshold.
- To do this, first we are going to define some thresholds in our script:

```
> pvalCutoff <- 0.01
> readsCutoff <- 10
> foldCutoff <- 1
```

- We will then use these to filter the data table, so that all results are significant (adjusted p-value < 0.01), above a certain threshold of normalized reads, and also only include complete cases (no 'Inf').

Thresholds and filtering incomplete cases

- We will do this in steps for clarity.
- First we remove the incomplete cases:

```
> resFiltered <- res[complete.cases(res) & res$baseMeanA != 0 &
res$baseMeanB != 0 & res$log2FoldChange!="-Inf"
& res$log2FoldChange!="Inf",]
```

- Then we filter for the reads count:

```
> resFiltered10Reads <-resFiltered[resFiltered$baseMeanA > readsCutoff &
resFiltered$baseMeanB > readsCutoff,]
```

- After this, we filter for our preferred p-value threshold:

```
resSig <- resFiltered10Reads[ resFiltered10Reads$padj < pvalCutoff, ]
```

- Finally, we also filter for a specific magnitude of fold change

```
> resFinal <- resSig[ abs(resSig$log2FoldChange) > foldCutoff, ]
```

Writing out and exploring results

5

Writing out results

- At this point, our data is ready to write to file. But first, lets make the column names easier to read:

```
> names(resFinal)[3] <- "meanWT"
> names(resFinal)[4] <- "meanPatient"
> names(resFinal)[6] <- "log2FoldChange (patient/WT)"
```

- We now have an easier table to read

```
> head(resFinal)
  id baseMean   meanWT meanPatient foldChange log2FoldChange (patient/WT)
     pval         padj
1  ENSG00000000003 133.6206 151.3947   115.84647  0.7651952
-0.3861002 1.441248e-03 7.606614e-03
7  ENSG00000000971  91.8473 163.9008    19.79379  0.1207669
-3.0497033 2.109916e-64 1.475070e-62
8  ENSG00000001036 401.8454 320.9894   482.70135  1.5037924
 0.5886054 2.138887e-12 3.145604e-11
```

Writing out results

- We can now write our data to file:

```
> write.table(resFinal, file="patient_vs_WT_p0_01.txt", sep="\t",
            row.names=TRUE, col.names=NA)
```

- This file is just a tab delimited text file, and can be opened in a text editor, Excel, etc., for further exploring results.

Exercise: DE analysis and exploration

- Go through the whole script and write out your final file of results. After applying all the filters, you should have **1263** significantly differentially expressed genes.
- Get just the gene identifiers (you can do this in R by filtering columns) and write them to file.
- Go to the following websites to explore your data:
    - Www.metabolicmine.org
    - bioinfo.vanderbilt.edu/webgestalt/

    At both of these, you will be asked to put in your gene list, and can then run some enrichment analysis.

- What types of genes do you notice in the dataset?