BCGES short courses, session 7, transcriptome sequencing (RNA-Seq)

Vincent Plagnol

Contents

1 GTF format to store information gene-centric information (20 minutes)

1.1 Ensembl data

If one works with genes and exons, it is important to have a format that captures this information. The file format that does this is the GTF format. A good place to download GTF file is the http://www.ensembl.org/info/data/ftp/index.html. One can start by using the `curl` function (which is a combination of `cat` and `url`) to obtain the first few lines of an example GTF file.

```
curl --silent ftp://ftp.ensembl.org/pub/release-76/gtf/homo_sapiens/Homo_sapiens.GRCh38.76.gtf.gz | \
   zcat | head -100 > results/human_gtf_example.gtf
```

Exercise: Go over the GTF format and understand what the fields mean, and how the data are organised.

You can now download the full ensembl file to get an idea of the size of the file. We will use the wget function that was used before in these practicals (note that the code below is not executed, because too long to go through).

```
wget -O results/ensembl_human_GRCh38.gtf.gz \
   ftp://ftp.ensembl.org/pub/release-76/gtf/homo_sapiens/Homo_sapiens.GRCh38.76.gtf.gz
```

1.2 UCSC data

UCSC is the other obvious place to obtain genome-scale data. The webpage you want to become familiar with is this one.

Exercise: Look for a human GTF file generally equivalent to the one you just downloaded from UCSC. Compare the sizes of both files, look for differences and similarities.

```
#You want to set the group option \textt{mRNA and EST}.
#Use the Human mRNA track
#In output format select \texttt{GTF - gene transfer format}
#Specify a name in the output file
#Maybe request a compressed file to limit transfer time
#Compressed the UCSC GTF is ... and the compressed Ensembl one is 16M.
```

2 Library normalization choices (40 minutes)

2.1 A toy example

A key issue for RNA-Seq data is normalization: how do you compare two RNA-Seq datasets generated at two different time points? One popular choice is RPKM which stands for reads per kb and per Million reads. The recipe is simple: divide by the total number of reads, divide by the length of the gene, and multiply by a number (one million, which is what M stands for) to get a number that is not too small. However, this may not be quite what you want. Indeed, we will know look at an example that illustrates the limitations of this measurement. Our dataset will be very basic (the code below is `R` code):

```
##let us assume that all genes have the same length, to put this problem aside for now
read.data <- data.frame (sample1 = 100, sample2 = c(rep(143, 70), rep(0, 30)))
## so one sample has all genes at level 100
##another has more reads for 70 genes and no read for the other 30
```

Exercise: Setting aside the gene length question, how would you compute RPKM values in this case? To get numbers easier to manipulate, let us rather us the number of reads per 1,000. How do you look into these RPKM values? Do they match what you would see as the intuitive explanation for these data?

```
RPKM.sample1 <- read.data$sample1/sum(read.data$sample1) * 1000
RPKM.sample2 <- read.data$sample2/sum(read.data$sample2) * 1000
table(RPKM.sample1)

## RPKM.sample1
##  10
## 100

table(RPKM.sample2)

## RPKM.sample2
##                0 14.2857142857143
##               30               70

## the tables above indicate that for sample1, all genes have the same level of
## expression (which happens to be 10).  However for sample 2, 70 genes have a 14,
## and 30 genes have a 0 value.  I don't think this matches the intuition we
## should have in this case. It seems to me that in that case one would rather
## conclude that 30 genes are not expressed in sample2, but that the other genes
## are pretty much on the same level.
```

So now what to do, and how to interpret the data. Here is an interesting forum answer that recapitulates the problem. I copy paste below the explanation:
To estimate the library size, simply taking the total number of (mapped or unmapped) reads is, in our experience, not a good idea. Sometimes, a few very strongly expressed genes are differentially expressed, and as they make up a good part of the total counts, they skew this number. After you divide by total counts, these few strongly expressed genes become equal, and the whole rest looks differentially expressed. The following simple alternative works much better:

- Construct a "reference sample" by taking, for each gene, the geometric mean of the counts in all samples.
- To get the sequencing depth of a sample relative to the reference, calculate for each gene the quotient of the counts in your sample divided by the counts of the reference sample. Now you have, for each gene, an estimate of the depth ratio.
- Simply take the median of all the quotients to get the relative depth of the library.

This is what the 'estimateSizeFactors' function of our DESeq package does.

This answer summarizes the problem well. One can see that in our case the 30 genes with 0 read have a large effect on the overall expression measurements that is probably not warranted. We will use the `DESeq` package to address this issue.

Exercise: There are three functions that you need to estimate the size factors in `DESeq` based on the dataset above. One is `newCountDataSet` to create a new object that `DESeq` can manipulate. The other two functions are: `estimateSizeFactors` and `sizeFactors` (the latter extracts the size factors from a `DESeq` object). Using these two functions (and starting with loading the `DESeq` library), can you compute the size factors, normalize the data using these, and get new gene level estimates of expression? Is the result now more consistent with your intutive interpretation of the data?

```
### Here is my answer to the question above, in R:
library(DESeq)
CDS <- newCountDataSet(read.data[, 1:2], condition = c('sample1', 'sample2'))
CDS <- estimateSizeFactors(CDS)
size.factors <- sizeFactors(CDS)
print(table(read.data$sample1 / size.factors[ 1 ]))

##
## 119.582607431014
##              100

print(table(read.data$sample2 / size.factors[ 2 ]))

##
##                0 119.582607431014
##               30               70
```

2.2 Further reading on normalization of RNA-Seq data

The issue of normalization of RNA-Seq data has been of interest to a lot of people. You can for example have a read of Lior Pachter's blog post on the matter. The blog links to a talk that is probably interesting (though I have not yet seen it).

This other blog post is a good read for the list of available methods, even though I do not think I agree with all the details. In particular the sentence: "Again, the methods in this section allow for comparison of features with different length WITHIN a sample but not BETWEEN samples" does not make sense to me. If we normalize, it is exactly to compare data across samples (there is no point otherwise). So while there are caveats, as always, I don't think this (rather crucial) statement makes sense.

3 Aligning RNA-Seq data and estimating gene expression levels (45 minutes)

Aligning short-read RNA-Seq data is not fundamentally different from aligning DNA sequencing data. It is however made more complex by the presence of introns, which can create reads or paired-reads spanning large distances. A popular aligner for RNA-Seq data is `tophat` and we will go over some basic commands.

3.1 Aligning with `tophat` and `bowtie`

It is important to note that the underlying alignment engine for `tophat` is `bowtie`, hence many commands are shared with standard calls to `bowtie`. We start by building a `bowtie` index for a short portion of chromosome 12, which we will use as an example for this class. Before you go through these steps, execute the script `scripts/tophat_bowtie_scripts.sh`. It will generate all the output files we want to look into, and the following goes through these commands in more details.

```
bowtie2-build -f ../data/RNASeq/chr12_short.fa ../data/RNASeq/chr12_short
```

With this, we can now perform the alignment step. But we first create some output folders to store all the output files:

```
mkdir results/tophat_output
```

Now we can start working with the fastq files:

```
f1=../data/RNASeq/reads_1.fq.gz
f2=../data/RNASeq/reads_2.fq.gz

tophat --no-coverage-search -o results/tophat_output -r 220 --library-type fr-unstranded \
      --segment-length 30 -G ../data/RNASeq/chr12_short.gtf ../data/RNASeq/chr12_short ${f1} ${f2}
```

3.2 Looking at the BAM file

Can you see what subtle differences can be found in a RNA-Seq BAM file compared to a DNA sequencing BAM file? The read extracted below should illustrate this.

```
samtools view results/tophat_output/accepted_hits.bam  | grep 16M2150N34M > results/odd_read.sam
```

3.3 `Cufflinks`

A popular software often associated with `tophat` is `cufflinks`. This piece of software is designed to estimate the abundance of each gene (and potentially isoforms). A call to `cufflinks` is pretty straightforward:

```
cufflinks  -o results/cufflinks_output --GTF ../data/RNASeq/chr12_short.gtf \
   results/tophat_output/accepted_hits.bam
```

4 Differential expression analysis (30 minutes)

We start by loading the DESeq package as well as an example dataset from a mouse brain RNA-Seq experiment.

```
library(DESeq)
load("../data/RNASeq/deseq_counts_TDP43.RData")
head(genes.counts)

##                    control_rep1_dexseq_counts.txt
## ENSMUSG00000000001                            208
## ENSMUSG00000000003                              0
## ENSMUSG00000000028                             15
## ENSMUSG00000000037                              9
## ENSMUSG00000000049                              4
## ENSMUSG00000000056                            233
##                    control_rep2_dexseq_counts.txt
## ENSMUSG00000000001                            295
## ENSMUSG00000000003                              0
## ENSMUSG00000000028                             26
## ENSMUSG00000000037                             20
## ENSMUSG00000000049                              1
## ENSMUSG00000000056                            390
##                    control_rep3_dexseq_counts.txt
## ENSMUSG00000000001                            239
## ENSMUSG00000000003                              0
## ENSMUSG00000000028                             13
## ENSMUSG00000000037                             13
## ENSMUSG00000000049                              3
## ENSMUSG00000000056                            346
##                    control_rep4_dexseq_counts.txt KD_rep1_dexseq_counts.txt
## ENSMUSG00000000001                            292                       326
## ENSMUSG00000000003                              0                         1
## ENSMUSG00000000028                             13                        21
## ENSMUSG00000000037                             21                        11
## ENSMUSG00000000049                              2                         2
## ENSMUSG00000000056                            381                       339
##                    KD_rep2_dexseq_counts.txt KD_rep3_dexseq_counts.txt
## ENSMUSG00000000001                       371                       316
## ENSMUSG00000000003                         0                         0
## ENSMUSG00000000028                        22                        18
## ENSMUSG00000000037                        12                        18
## ENSMUSG00000000049                         1                         1
## ENSMUSG00000000056                       359                       317
##                    KD_rep4_dexseq_counts.txt
## ENSMUSG00000000001                       339
## ENSMUSG00000000003                         0
## ENSMUSG00000000028                        18
## ENSMUSG00000000037                        30
## ENSMUSG00000000049                         2
## ENSMUSG00000000056                       379
```

We can now define the model for the differential expression analysis:

```
formula1 <- count ~ condition
formula0 <- count ~ 1
design.deseq <- c('control', 'control', 'control', 'control', 'KD', 'KD', 'KD', 'KD')
```

And now the computations can properly start. Note that these steps are very long, and therefore the code is not executed as part of this file (to be more precise, it is executed once, and the output is saved).

```r
CDS <- newCountDataSet(genes.counts, condition = design.deseq)

CDS <- estimateSizeFactors(CDS)
CDS <- estimateDispersions(CDS, method = 'pooled')

fit0 <- fitNbinomGLMs( CDS, formula0 )
fit1 <- fitNbinomGLMs( CDS, formula1 )

deseq.pval <- fit1
deseq.pval$EnsemblID <- row.names( deseq.pval)
deseq.pval$basic.pval <- signif(nbinomGLMTest( fit1, fit0 ), 4)
save(list = 'deseq.pval', file = 'results/DE_pvalues_ranked.RData')
```

See below some polishing: a multiple testing/false discovery rate Bonferroni-Hochberg analysis, and the ordering of the results by significance of P-values.

```r
load('results/DE_pvalues_ranked.RData')
deseq.pval$adj.pval <- signif(p.adjust( deseq.pval$basic.pval, method="BH" ), 4)

deseq.pval <- deseq.pval[ order(deseq.pval$basic.pval, decreasing = FALSE), ]
head(deseq.pval)

##                      (Intercept) conditionKD deviance converged
## ENSMUSG00000023224       5.788       1.733    4.110      TRUE
## ENSMUSG00000023826       6.559      -2.002    7.912      TRUE
## ENSMUSG00000026547       6.032       1.688    3.296      TRUE
## ENSMUSG00000039419       9.698      -1.185   12.093      TRUE
## ENSMUSG00000040424       8.450      -1.373    6.263      TRUE
## ENSMUSG00000041459      10.080      -1.691    5.526      TRUE
##                              EnsemblID basic.pval adj.pval
## ENSMUSG00000023224 ENSMUSG00000023224          0        0
## ENSMUSG00000023826 ENSMUSG00000023826          0        0
## ENSMUSG00000026547 ENSMUSG00000026547          0        0
## ENSMUSG00000039419 ENSMUSG00000039419          0        0
## ENSMUSG00000040424 ENSMUSG00000040424          0        0
## ENSMUSG00000041459 ENSMUSG00000041459          0        0
```