Homework: Analyzing RNA-Seq Data with edgeR and DESeq2

J. Jedediah Smith

2023-04-16

## From Reads to Count Matrices

Let us tabulate, how often a read matches to a gene region. Let there be 2 rows for the 2 genes. And let there be for n students be n \* 2 columns (n columns for control and n columns for “case”). In this matrix we enter integer values: for each gene, each students reports two numbers: the number of reads that match to that gene from the normal group and the control group. This is called the “count matrix”.

## Read Data

Adjust the variable MY\_DATA\_PATH so it corresponds to the directory where the inputfiles for this work (‘mock\_case\_1.txt’, ‘mock\_case\_2.txt’ , etc.) are located.

MY\_DATA\_PATH = 'mockgenome2' # CHANGE IF NEEDED  
indir = MY\_DATA\_PATH  
stopifnot(file.exists(indir))  
genomefile = 'mock\_genome.txt'  
sequencefile = 'mock\_sequences.txt'  
casefiles = c('mock\_case\_1.txt', 'mock\_case\_2.txt',  
 'mock\_case\_3.txt', 'mock\_case\_4.txt')  
controlfiles = c('mock\_control\_1.txt', 'mock\_control\_2.txt',  
 'mock\_control\_3.txt')  
genome = readLines(file.path(indir, genomefile))  
sequences = readLines(file.path(indir, sequencefile))

The read genome sequence is: TTTTAATCAGCTTCCGGTGACATATATAAATGTTTTCTCGGGAGTAGGACTGGTGGGCGGGGCATATAAATCTTGGTTATTATTT The individual sequences are:

print(sequences)

## [1] "TTTTAATCAG" "CTTCCGGTGACATATATAAA"   
## [3] "TGTTTTCTCG" "GGAGTAGGACTGGTGGGCGGGGCATATAAA"  
## [5] "TCTTGGTTATTATTT"

## Exercise 1: Find Matches using matchPattern

Use the control sequences to find matching regions. Identify the region where the first sequence from mock\_control\_1.txt is matching the mock genome sequence. Use the function matchPattern from the Biostrings package.

library(Biostrings)  
  
gx <- DNAString(genome)  
cr1 <- readLines(file.path(indir, controlfiles[1]))  
  
matches <- matchPattern(cr1[1],gx)  
matches

## Views on a 85-letter DNAString subject  
## subject: TTTTAATCAGCTTCCGGTGACATATATAAATGTT...GGTGGGCGGGGCATATAAATCTTGGTTATTATTT  
## views:  
## start end width  
## [1] 18 25 8 [TGACATAT]

## Exercise 2: Find Matches using matchPDict

A more elegant way to obtain all matches for a set of patterns is using the functions PDict and matchPDict that first create a dictionary of patterns and then perform a fast search in a target region.

1. Use the function PDict to generate a dictionary of patterns for all sequences from file mock\_control\_1.txt

cr1d <- PDict(cr1)  
cr1d

## TB\_PDict object of length 39 and width 8 (preprocessing algo="ACtree2")

1. Use function matchPDict to generate a list of all matches of all sequences with respect to the mock genome. Apply function unlist to the result of matchPDict to obtain an easier to process result. Store the result in a variable called my\_ranges.

my\_ranges <- unlist(matchPDict(cr1d,gx))  
my\_ranges

## IRanges object with 39 ranges and 0 metadata columns:  
## start end width  
## <integer> <integer> <integer>  
## [1] 18 25 8  
## [2] 14 21 8  
## [3] 11 18 8  
## [4] 11 18 8  
## [5] 15 22 8  
## ... ... ... ...  
## [35] 47 54 8  
## [36] 54 61 8  
## [37] 51 58 8  
## [38] 45 52 8  
## [39] 43 50 8

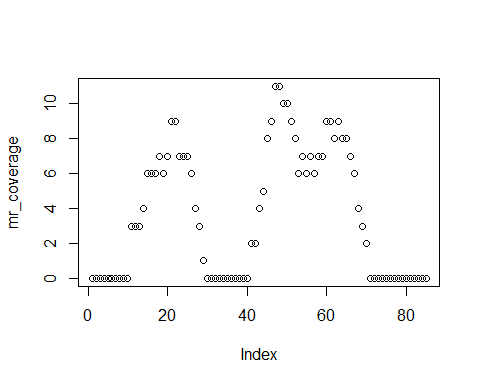
1. Use the function coverage on the result of the previous step to find the coverage of of the matches (in other words for each position on the mock genome a number that indicates how many mock reads are matching). Print the resulting output. Hint: The function coverage will not show the last region of the mock genome without any matches. One can nudge the function to also include that region with the parameter width and provided the length of the target sequence.

mr\_coverage <- coverage(my\_ranges, width = length(gx))  
mr\_coverage

## integer-Rle of length 85 with 39 runs  
## Lengths: 10 3 1 3 1 1 1 2 3 1 1 ... 2 1 1 2 1 1 1 1 1 15  
## Values : 0 3 4 6 7 6 7 9 7 6 4 ... 9 8 9 8 7 6 4 3 2 0

1. Plot the coverage using function plot as line-chart. Hint: First convert the run-length encoded result from the previous step to a regular vector using the function as.vector(…)

# Convert length to vector  
mr\_coverage@lengths <- as.vector(mr\_coverage@lengths)  
  
# Plot coverage  
plot(mr\_coverage)



1. Identify gene regions: inspect the coverage plot. How many (mock) genes do you think are encoded in the mock genome? What genomic regions to they correspond to?

# There seems to be 2 large peaks. I would imagine each of them represents a mock gene region. The first goes from ~10 to ~30. The second from ~40 to ~70.

## Exercise 3: From Read Alignments to a Count Matrix

Let’s write code that counts how many mock reads are matching a particular region in our mock genome. We already have the key information in form of the result of function matchPDict above.

Let’s create to IRanges objects that represent the intervals of the genes we identified:

library(IRanges)  
g1range = IRanges(start=11, end=30)  
g2range = IRanges(start=41, end=70)

With function findOverlaps from R package IRanges we can know identify all reads that overlap with the genomic regions corresponding to gene1 and gene2 like so:

# variable my\_ranges contains result from matchPDict above  
findOverlaps(g1range, my\_ranges)

## Hits object with 13 hits and 0 metadata columns:  
## queryHits subjectHits  
## <integer> <integer>  
## [1] 1 1  
## [2] 1 2  
## [3] 1 3  
## [4] 1 4  
## [5] 1 5  
## ... ... ...  
## [9] 1 9  
## [10] 1 10  
## [11] 1 11  
## [12] 1 12  
## [13] 1 13  
## -------  
## queryLength: 1 / subjectLength: 39

1. Instead of function findOverlaps, use similar function countOverlaps to count how many mock reads are overlapping with g1range and g2range. What are the 2 counts?

# Count overlaps for each region  
countOverlaps(g1range, my\_ranges) # 13

## [1] 13

countOverlaps(g2range, my\_ranges) # 26

## [1] 26

*Finally*, we identified two entries we can use in our count matrix!

1. Write code, that generates a complete count matrix for all 7 sample files and the two genes, resulting in a matrix with 2 rows and 7 columns. It strongly encouraged to define your own functions so that you do not have to repeat similar code 2x7=14 times! Print the 2x7 count matrix you computed.

# Function to count which reads overlap with given gene range.  
GeneCount <- function(g\_range,r\_range) {  
 return(countOverlaps(g\_range, r\_range))  
}  
  
# Function to grab reads from designated file.  
ReadGrab <- function(set,index) {  
 cr <- readLines(file.path(indir, set[index]))  
 crd <- PDict(cr)  
 return(unlist(matchPDict(crd,gx)))  
}  
  
# Create the matrix by utilizing the functions.  
count\_matrix <- data.frame(cbind(  
 Gene1 = c(  
 GeneCount(g1range,ReadGrab(casefiles,1)),  
 GeneCount(g1range,ReadGrab(casefiles,2)),  
 GeneCount(g1range,ReadGrab(casefiles,3)),  
 GeneCount(g1range,ReadGrab(casefiles,4)),  
 GeneCount(g1range,ReadGrab(controlfiles,1)),  
 GeneCount(g1range,ReadGrab(controlfiles,2)),  
 GeneCount(g1range,ReadGrab(controlfiles,3))  
 ),  
 Gene2 = c(  
 GeneCount(g2range,ReadGrab(casefiles,1)),  
 GeneCount(g2range,ReadGrab(casefiles,2)),  
 GeneCount(g2range,ReadGrab(casefiles,3)),  
 GeneCount(g2range,ReadGrab(casefiles,4)),  
 GeneCount(g2range,ReadGrab(controlfiles,1)),  
 GeneCount(g2range,ReadGrab(controlfiles,2)),  
 GeneCount(g2range,ReadGrab(controlfiles,3))  
 )  
))  
  
# Make sure to name our rows!  
rownames(count\_matrix) <- c("Case1", "Case2", "Case3", "Case4",  
 "Control1", "Control2", "Control3")  
  
# Print the matrix  
count\_matrix

## Gene1 Gene2  
## Case1 14 28  
## Case2 18 36  
## Case3 22 44  
## Case4 26 52  
## Control1 13 26  
## Control2 16 32  
## Control3 19 38

## Using edgeR for RNA-Seq Normalization

Now we will use the edgeR package to use counts of RNA-Seq reads aligned to different gene regions in order to estimate differential gene expression. This is mock data, we assume that we have 2 experimental conditions called ‘Case’ and ‘Control’.

library(edgeR)  
# library(dplyr)  
options(warn=-1)  
counts = matrix(0, nrow=2, ncol=7) # 0 is just placeholder, will be replaced  
rownames(counts) = c('Gene1', 'Gene2')  
colnames(counts) = c("Case1", 'Case2', 'Case3', 'Case4',  
 'Control1', 'Control2', 'Control3')  
counts['Gene1',] = c(17, 12, 10, 5, 85, 67, 90) # counts for Gene1  
counts['Gene2',] = c(72, 81, 79, 92, 36, 41, 39) # counts for Gene2  
knitr::kable(counts)

|  | Case1 | Case2 | Case3 | Case4 | Control1 | Control2 | Control3 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Gene1 | 17 | 12 | 10 | 5 | 85 | 67 | 90 |
| Gene2 | 72 | 81 | 79 | 92 | 36 | 41 | 39 |

df = tibble::tibble(ID=c("Case1",'Case2','Case3','Case4',  
 'Control1', 'Control2', 'Control3'),  
 Group=c('Case','Case','Case','Case',  
 'Control','Control','Control'))  
knitr::kable(df)

| ID | Group |
| --- | --- |
| Case1 | Case |
| Case2 | Case |
| Case3 | Case |
| Case4 | Case |
| Control1 | Control |
| Control2 | Control |
| Control3 | Control |

### Grouping of Experiments

We have to tell the algorithm how experiments are grouped into several groups. Examples are ‘tumor’ versus ‘normal’ or ‘treatment’ versus ‘control’. In our case we call it ‘Case’ versus ‘Control’.

grouping <- factor(df[['Group']])  
print(grouping)

## [1] Case Case Case Case Control Control Control  
## Levels: Case Control

### Creating a Differential Gene Expression List Object (DGEList)

count\_dge <- edgeR::DGEList(counts=counts,group=grouping)  
count\_dge

## An object of class "DGEList"  
## $counts  
## Case1 Case2 Case3 Case4 Control1 Control2 Control3  
## Gene1 17 12 10 5 85 67 90  
## Gene2 72 81 79 92 36 41 39  
##   
## $samples  
## group lib.size norm.factors  
## Case1 Case 89 1  
## Case2 Case 93 1  
## Case3 Case 89 1  
## Case4 Case 97 1  
## Control1 Control 121 1  
## Control2 Control 108 1  
## Control3 Control 129 1

### Using edgeR

design <- model.matrix(~ grouping)  
eset\_dge <- edgeR::estimateDisp(count\_dge, design)  
fit <- edgeR::glmQLFit(eset\_dge, design)  
result <- edgeR::glmQLFTest(fit, coef=2)

### Viewing edgeR Results

knitr::kable(topTags(result))

|  | logFC | logCPM | F | PValue | FDR |
| --- | --- | --- | --- | --- | --- |
| Gene2 | -1.438682 | 19.20025 | 121.5081 | 6e-07 | 7e-07 |
| Gene1 | 2.487220 | 18.60141 | 118.8965 | 7e-07 | 7e-07 |

| x |
| --- |
| BH |

| x |
| --- |
| groupingControl |

| x |
| --- |
| glm |

## Exercise 4: Fixing a Terrible Mistake

Inspect the initial results table returned by edgeR and function topTags.

1. According to the initial results, what genes are up-regulated? Write your result without using code

# Gene 1 is up-regulated on account of its positive logFC.

1. According to the initial results, genes are down-regulated? Write your result without using code

# Gene 2 is down-regulated on account of its negative logFC.

1. Compare this with what you expect by inspecting the initial count matrix. Does this edgeR result make sense? Why not?

# If Gene 1 was up-regulated and Gene 2 down-regulated, we'd expect to see larger counts for Gene 1 than Gene 2. However, our data shows the opposite. Gene 2 has higher counts than Gene 1. It doesn't make sense.

1. How can this have happened? How can we fix it? Hint: although the initial results are completely wrong, the mistake may not be so easy to find (‘deep bug’). The answer lies in the way the factor variable ‘grouping’ is ordered. edgeR looks at the order of the provided factor variable for experiment groups and uses it to define up and down-regulation. You can over-ride the default (alphabetical) order of factors in R function factor used above by specifically specifying the order of the factors using the parameter levels.

# It appears that Edge R is defining up and down regulation based on the experimental data because it appears first in the list. This is not what we want. It should be defined based on the control data. We need to reorder the data so that the control groups come first.

1. Implement the fix you identified in the previous step and rerun this markdown file. Confirm that the edgeR results are now more in line with what we expect by inspecting the count data.

# Reorder our groups  
grouping <- factor(df[['Group']], levels = c("Control","Case"))  
print(grouping)

## [1] Case Case Case Case Control Control Control  
## Levels: Control Case

# Reorder our data  
count\_dge <- edgeR::DGEList(counts=counts,group=grouping)  
count\_dge

## An object of class "DGEList"  
## $counts  
## Case1 Case2 Case3 Case4 Control1 Control2 Control3  
## Gene1 17 12 10 5 85 67 90  
## Gene2 72 81 79 92 36 41 39  
##   
## $samples  
## group lib.size norm.factors  
## Case1 Case 89 1  
## Case2 Case 93 1  
## Case3 Case 89 1  
## Case4 Case 97 1  
## Control1 Control 121 1  
## Control2 Control 108 1  
## Control3 Control 129 1

# Rerun the Edge R stuff  
design <- model.matrix(~ grouping)  
eset\_dge <- edgeR::estimateDisp(count\_dge, design)  
fit <- edgeR::glmQLFit(eset\_dge, design)  
result <- edgeR::glmQLFTest(fit, coef=2)  
  
# Make our table  
knitr::kable(topTags(result)) # Results now flip which is up or down regulated and make more sense.

|  | logFC | logCPM | F | PValue | FDR |
| --- | --- | --- | --- | --- | --- |
| Gene2 | 1.438682 | 19.20025 | 121.5081 | 6e-07 | 7e-07 |
| Gene1 | -2.487220 | 18.60141 | 118.8965 | 7e-07 | 7e-07 |

| x |
| --- |
| BH |

| x |
| --- |
| groupingCase |

| x |
| --- |
| glm |

1. What work habits can help to prevent or identify code bugs that dramatically impact results?

# Always checking to make sure that data makes sense with what one expects. If we took the Edge R results at face value and did not compare to our counts, then we wouldn't have noticed our mistake.

## Exercise 5: Inspecting the Fixed edgeR Result

Inspect the results table returned by edgeR and function topTags after you applied the code-fix.

1. What genes are up-regulated?

# Gene 2 is up-regulated on account of its positive logFC.

1. What genes are down-regulated?

# Gene 1 is down-regulated on account of its negative logFC.

1. What is the ratio of the mean expression of ‘Case’ experiments divided by mean expression of ‘Control’ for Gene1? Hint: logFC stands for the logarithm (base 2) of the fold change between

# Fold change is our ratio between case and control.  
# All we need to do is undo log2.  
  
2^(-2.487220)

## [1] 0.1783496

# Our ratio is 0.1783496.

## Using DESeq2 instead of edgeR

Now we will apply a different package for analyzing RNA-Seq data called DESeq2. The DESeq2 approach is based on the negative binomial distribution. Both the negative binomial distribution and the Poisson distribution are important tools for modeling count data, but they have different applications and assumptions. The Poisson distribution is appropriate when modeling events that occur at a fixed rate, while the negative binomial distribution is better suited for modeling events that occur randomly and exhibit overdispersion.

One key difference between the two packages is that DESeq2 assumes that the variance and mean of the count data are related by a negative binomial distribution, while edgeR uses a more flexible approach that allows for gene-specific dispersion estimates. This can make edgeR more suitable for datasets with high levels of biological variability or low sequencing depth.

Another difference is that DESeq2 is better suited for datasets with a large number of samples, while edgeR may be more appropriate for datasets with fewer samples. DESeq2 can also handle complex experimental designs, such as those involving multiple factors or time series data.

# BiocManager::install('DESeq2')  
library("DESeq2")

The Interface of DESeq2 is slightly different compared to edgeR. As before we need to provide information about how the differernt experiments are grouped, but instead of a factor variable we have to provide a data frame, with the rownames corresponding to the experiment ids (equal to column names of the count data matrix) and one column containing the group-id name of the individual experiment.

rownames(df) = df$ID  
knitr::kable(df)

| ID | Group |
| --- | --- |
| Case1 | Case |
| Case2 | Case |
| Case3 | Case |
| Case4 | Case |
| Control1 | Control |
| Control2 | Control |
| Control3 | Control |

dds <- DESeqDataSetFromMatrix(countData = counts,  
 colData = df,  
 design = ~ Group)

### Perform Computation

dds <- DESeq(dds)  
print(dds)

## class: DESeqDataSet   
## dim: 2 7   
## metadata(1): version  
## assays(4): counts mu H cooks  
## rownames(2): Gene1 Gene2  
## rowData names(22): baseMean baseVar ... deviance maxCooks  
## colnames(7): Case1 Case2 ... Control2 Control3  
## colData names(3): ID Group sizeFactor

## Exercise 6: Extract DESeq2 Results

1. apply the function results from package DESeq2 to the result object (variable dds) created above. That function needs as first argument the result object, and as second argument (called ‘contrast’) a character vector of length 3 with

* first element being the column name containing the experiment group name,
* the second element containing the group name corresponding to the treatment or case, and
* the third element the character string corresponding to the control experiment group name. Print the resulting data frame.

contrast <- c("Group","Case","Control")  
dds\_results <- results(dds, contrast)  
dds\_results

## log2 fold change (MLE): Group Case vs Control   
## Wald test p-value: Group Case vs Control   
## DataFrame with 2 rows and 6 columns  
## baseMean log2FoldChange lfcSE stat pvalue padj  
## <numeric> <numeric> <numeric> <numeric> <numeric> <numeric>  
## Gene1 31.3937 -1.94404 0.329034 -5.90832 3.45619e-09 3.45619e-09  
## Gene2 75.0338 2.06329 0.296106 6.96809 3.21281e-12 6.42563e-12

1. Post-process results. We want to filter out result rows where the result is undefined (“NA”) plus we want to sort by statistical significance. When given the choice between a raw P-value and a P-value that has been adjusted for multiple-testing correction, one should choose the adjusted version. This column is called Padj in case of the DESeq2 package and FDR (false discovery rate) in case of the edgeR package.

# Remove any NA data. (There is none?)  
dds\_results <- na.omit(dds\_results)  
  
# Sort by padj  
dds\_results <- dds\_results[order(dds\_results$padj),]  
  
# Print  
dds\_results

## log2 fold change (MLE): Group Case vs Control   
## Wald test p-value: Group Case vs Control   
## DataFrame with 2 rows and 6 columns  
## baseMean log2FoldChange lfcSE stat pvalue padj  
## <numeric> <numeric> <numeric> <numeric> <numeric> <numeric>  
## Gene2 75.0338 2.06329 0.296106 6.96809 3.21281e-12 6.42563e-12  
## Gene1 31.3937 -1.94404 0.329034 -5.90832 3.45619e-09 3.45619e-09

1. By what factor are genes Gene1 and Gene2 up or down-regulated?

# Fold change is our ratio between case and control.  
# All we need to do is undo log2.  
  
2^(-1.94404) # Gene 1, factor of 0.25.

## [1] 0.2598877

2^(2.06329) # Gene 2, factor of 4.

## [1] 4.179383

1. Compare the results from edgeR and DESeq2. Both are respected approaches for RNA-Seq data normalization and analysis. Are the results very similar? What does that mean in practice about your data analysis?

# Here are the results from Edge R.  
  
2^(-2.487220) # Gene 1, factor of 0.17.

## [1] 0.1783496

2^(1.438682) # Gene 2, factor of 2.7.

## [1] 2.710731

# When compared with the results from the DESeq, they are substantially different. In practice, this means the approach taken may matter significantly. We should try to understand the differences between approaches choose what is best based on the data and goals of the study.