An RNA-seq Data Analysis Protocol

David Chisanga and Wei Shi

2021-02-08

Contents

Preface

This tutorial describes a RNA-seq analysis protocol that can be used to analyse Bulk RNA-seq data on the Bioinformatics server. The guide provides a brief discussion and examples of the steps involved in the analysis of Bulk RNA-seq data, from raw data through to differential expression analysis.

Chapter 1

Overview

Discovering differentially expressed genes is one of the most important applications of RNA sequenicng technologies. This RNA-seq analysis protocol provides step-by-step procedures for a successful analysis of an RNA-seq dataset. The protocol can be applied to RNA-seq data generated from any species that has a reference genome and an annotation of genes and it can be applied to both small and large RNA-seq experiments.

This protocol comprises three major steps including read mapping, read counting and statistical testing (Fig. 1). Read mapping and counting are performed using Rsubread package (Liao et al., 2019) and statistical testing is carried out using limma package (Ritchie et al., 2015).

Read mapping.
Read mapping is the first step in an RNA-seq analysis. In this step, an index for the reference genome will be first built using *buildindex()* function and then reads will be aligned to the genome using the *align()* function (Liao et al., 2019, 2013b). Index building is a one-off operation. The built index can be reused in future RNA-seq analyses. Read mapping results generated from *align()* are saved to BAM or SAM format files. *align()* also returns a mapping summary including percentages of mapped reads, uniquely mapped reads and multi-mapping reads.

Read counting
After read mapping is completed, mapped reads can then be counted to genes using the *featureCounts()* function (Liao et al., 2013a). Other than read mapping data, a gene annotation must be provided to *featureCounts()* for read counting. Rsubread package includes inbuilt gene annotations for human and mouse genomes. These annotations were constructed based on the NCBI RefSeq gene annotation database (O'Leary et al., 2015). Gene annotations for other species can be downloaded from NCBI RefSeq or other databases. *featureCounts()* (Shi et al., 2020) returns a count table which includes read counts for each gene in each library. It also returns other information such as counting summary for

each library and gene length.

Normalization
After read counts are generated, the *voom()* function in limma will be used to convert raw counts to counts per million (CPM), estimate the mean-variance relationship and compute observation levelweights (Law et al., 2014). The quantile normalization will also be applied to the data. Linear models are then fitted to the normalized data using *lmFit()* function. Empirical Bayes moderated t-statistic will be used to test the statistical significane of gene expression changes (Smyth, 2004). This test is performed using the *eBayes()* function in limma (Smyth et al., 2020). The output of statistical testing will include differentially expressed genes if any are found.

Chapter 2

Prerequisites

Before you get started with the rest of the analysis, it is important that you have the necessary data and software that will be used in this analysis.

2.1 Data

RNA-seq data
An RNA-seq dataset generated in a published study (Delconte et al., 2016) is used as an example dataset in this protocol.

Two samples (wild-type and Cish-/- natural killer cells) are included in this dataset and each sample has two biological replicates. These data were already deposited in the Gene Expression Omnibus database (GSE79409). However for the convenience of this analysis,FASTQ files of the raw RNA-seq data were also saved in the 'Workshop_RNAseq' directory on Z drive.

A text file called "Targets.txt", which includes relevant sample information, can also be found in this directory.

Reference genome data

A FASTA-format file including all chromosomal sequences of the GRCm38/mm10 genome was also saved in 'Workshop_RNAseq' directory on Z drive.

2.2 SOFTWARE

The following software tools should be installed on a UNIX server and on your laptop:

- R (https://www.r-project.org)
- Rsubread (http://bioconductor.org/packages/release/bioc/html/Rsubread.html)
- limma (http://bioconductor.org/packages/release/bioc/html/limma.html)
- edgeR (http://bioconductor.org/packages/release/bioc/html/edgeR.html)
- org.Mm.eg.db (http://bioconductor.org/packages/release/data/annotation/html/org.Mm.eg.db.html)
- statmod (https://CRAN.R-project.org/package=statmod)

Consult the R Project website for the installation of R (https://www.r-project.org/) (R Core Team, 2020). Make sure the latest release version of R is downloaded and installed. After R is installed, launch R and type the following commands to install *Rsubread*, *limma*, *edgeR*, *org.Mm.eg.db* and *statmod*:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
install.packages("BiocManager")

BiocManager::install(c("Rsubread","limma","edgeR","org.Mm.eg.db"),update = T)

if(!requireNamespace("statmod",quietly=TRUE))
  install.packages("statmod")
```

Alternatively, you may use Rstudio to run this protocol.

Chapter 3

Running Environment

3.1 RStudio

This is the recommended approach for running this protocol. An RStudio application ('RStudio(2672)') has been created under the 'Analysis' tab in the Remote Access Facility. After you log into it (using your email account name and password), change to the directory '/data/RawPrimary/Public' (you can do this by choosing Session > Set Working Directory >Choose Directory and then using the '…' to specify the directory you want to change to) and then you will find a folder called 'Workshop_RNAseq' which includes all the materials included in this Workshop. You can make a copy of this folder and then start to run the protocol in your own folder.

3.2 UNIX server + laptop

Run read mapping and counting on a UNIX server and then perform the rest of the analysis on your laptop. Refer to the document "How to Access Linux BioInformatics Analysis Platform.pdf" for how to access the Bioinformatics UNIX server. Once you logged in the server, you can issue the following commands to copy the Workshop data to a directory you create on the server and to launch R. You are then ready to run the protocol.

```
# connect AllStaffShare Drive to the server
cifscreds add svr-fs95
```

```
# create your own directory, eg 'my_directory'
cd /data/Processing/Public/
mkdir my_directory
```

```
# copy the Workshop data to your directory
cp -r /mnt/AllStaffShare/Workshop_RNAseq my_directory
cd my_directory/Workshop_RNAseq
```

```
# load R
module load R-bundle-Bioconductor
R
```

3.3 Laptop only

Run your entire RNA-seq analysis on your laptop. If you choose this option, you should build a low-memory index (split index) for the reference genome before performing read mapping.

Chapter 4

RNA-SEQ analysis protocol

Please note that your analysis results might be slightly different from those shown in this protocol due to software version differences.

We first change the working directory to the Workshop folder

```r
setwd("Workshop_RNAseq/")
```

Build index for a reference genome

Step 1
Start an R session and build an index for the reference genome *GRCm38/mm10*. The created index files will be saved to the current working directory. This index only needs to be built once and it can be reused in future RNA-seq data analyses.
TIMING ~40 mins

```r
library(Rsubread)
```

```r
buildindex("mm10_reference", "mm10.fa")
```

If you build an index on a laptop, it is recommended to build a split index. The amount of requested memory should be roughly half of the total memory available on your laptop. For example, if your laptop has 8GB memory, you may use the following command to build a split index with 4GB memory usage.
TIMING ~90 mins

```r
buildindex("mm10_reference",
           "mm10.fa",
           indexSplit = TRUE,
           memory = 4000)
```

Create sample-related information and evaluate quality of sequencing TIMING ~ 3 mins

Step 2

Create a tab-delimited text file that contains sample-related information such as FASTQ file names, sample names and cell types. For convenience, a file called "Targets.txt" has already been created for this dataset and we read in this file:

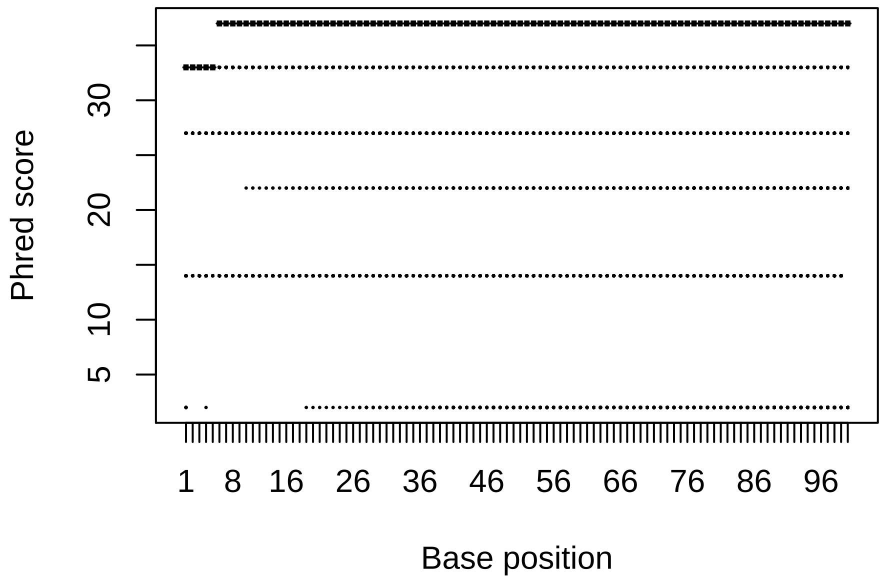
```
library(limma)
targets <- readTargets("Targets.txt")
targets
```

```
##                                    Library   Sample CellType
## 1 CIS_1_C6AG6ANXX_ATCACG_L008_R1.fastq.gz CIS_rep1      CIS
## 2 CIS_2_C6AG6ANXX_CGATGT_L008_R1.fastq.gz CIS_rep2      CIS
## 3  WT_3_C6AG6ANXX_TTAGGC_L008_R1.fastq.gz  WT_rep1       WT
## 4  WT_4_C6AG6ANXX_TGACCA_L008_R1.fastq.gz  WT_rep2       WT
```

Step 3
Choose one of the libraries and examine the Phred quality scores of read bases at each read position. Boxplots of Phred scores are generated for each base position.

```
qs <- qualityScores(targets$Library[1], offset = 33)
```

```
boxplot(qs,
        xlab = "Base position",
        ylab = "Phred score",
        cex = 0.1)
```



Align and count reads TIMING ~6 min per library

Step 4
Map sequence reads to mouse genome GRCm38/mm10. It is recommended to provide gene annotation to the read mapping. Mapped reads are saved to BAM files and a mapping summary is returned to R after read mapping is completed.

```
align.outfiles <- paste(targets$Sample, "bam", sep = ".")
align.summary <-
  align(
    "mm10_reference",
    targets$Library,
    output_file = align.outfiles,
    nthreads = 10,
    useAnnotation = TRUE,
    annot.inbuilt = "mm10"
  )
align.summary
```

Step 5
Assign reads to overlapping genes and generate an R object that contains a count table, gene annotation and counting summary. The count table contains number of reads assigned to each gene in each library. The gene annotation includes Entrez gene identifier, chromosomal coordinates of gene exons and gene length (total number of non-overlapping exonic bases each gene has). The counting summary gives the number of successfully assigned reads in each library and also numbers of reads that failed to be assigned due to filtering. The ‘Status’ column

in this summary includes a ‘Assigned’ category and also multiple ‘Unassigned’ categories corresponding to different read filters used in counting.

```
counts.gene <- featureCounts(align.outfiles, "mm10", nthreads = 10)
counts.gene$stat
```

Save R data and switch from the UNIX server to a personal computer TIMING < 1 min

(Steps 6 and 7 can be skipped if you run the protocol using ‘RStudio (2672)’ or laptop only)

Step 6 (optional)

Save all generated R objects to a file and then copy the file to your personal computer.

```
save.image("Counts.rdata")
```

Step 7 (optional)

Launch R on your personal computer and load R objects from the copied file.

```
load("Counts.rdata")
```

Perform differential expression analysis TIMING < 1 min

Step 8

Load required libraries.

```
library(limma)
library(edgeR)
library(statmod)
suppressMessages(library(org.Mm.eg.db))
```

Step 9

Create a design matrix.

```
ct <- factor(targets$CellType)
design <- model.matrix(~0 + ct)
colnames(design) <- levels(ct)
```

Step 10

Map Entrez gene identifiers to gene symbols and create an R object containing annotation data for each gene. The annotation data include Entrez gene identifier, gene symbol and gene length.

```
tmp <- org.Mm.egSYMBOL
entrez_symbol <- as.list(tmp[mappedkeys(tmp)])
entrez_symbol <- sapply(entrez_symbol, function(x)
  x[1])
genes <- counts.gene$annotation$GeneID
m <- match(genes, names(entrez_symbol))
genes <- data.frame(
  EntrezID = genes,
  Symbol = entrez_symbol[m],
  Length = counts.gene$annotation$Length,
  stringsAsFactors = FALSE
)
```

Step 11

Remove from analysis those genes that did not express, or expressed at a very low level, in all cell types. Here we require a gene to have a CPM value greater than 0.5 in at least two libraries to be included in the subsequent analysis. The reason we require at least two libraries is because there are two biological replicates generated for each sample in this dataset.

```
keep <- rowSums(cpm(counts.gene$counts) > 0.5) >= 2
sum(keep)
```

```
## [1] 11969
```

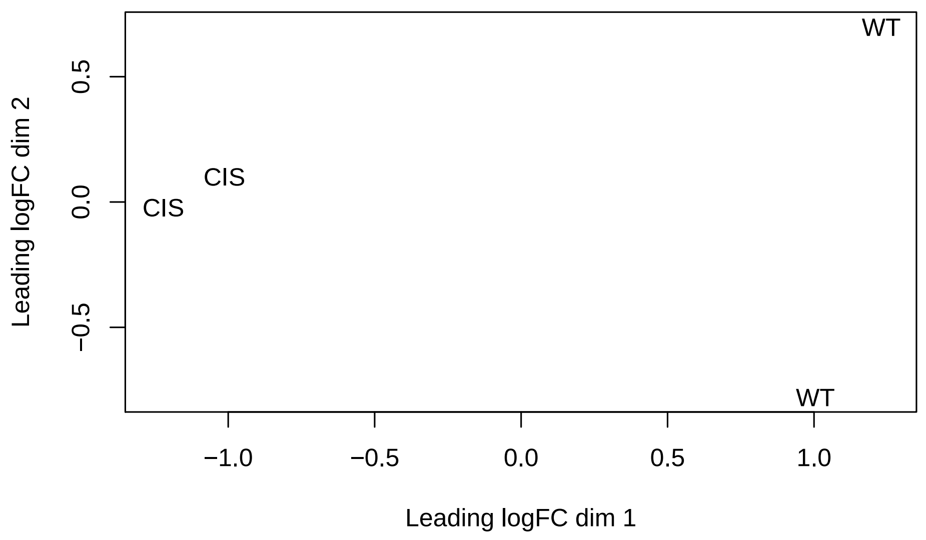
Step 12

Transform count data to log2-cpm, estimate the mean-variance relationship and compute observation-level weights. Log2-cpm expression values of genes are then normalized using the quantile method and converted to log2-rpkm (log2 reads per kilobases of exons per million reads) values.

```
y <-
  voom(counts.gene$counts[keep, ], design, normalize.method = "quantile")
y$genes <- genes[keep, ]
y$E <- y$E - log2(y$genes$Length / 1000)
```

Step 13
Cluster samples via multidimensional scaling.

```
plotMDS(y, labels=targets$CellType)
```



Step 14

Fit linear models to genes and perform statistical testing to discover differentially expressed genes. A contrast matrix is constructed to specify the comparisons between samples. The empirical Bayes moderated t-statistic is used to assess the differential expression of genes. A FDR (false discovery rate) cutoff is applied for calling differentially expressed genes. For this analysis, an FDR cutoff of 0.05 was applied (default cutoff value in decideTests()).

```
fit <- lmFit(y, design)
contr <- makeContrasts(CISvsWT = CIS - WT, levels = design)
fit.contr <- eBayes(contrasts.fit(fit, contr))
dt <- decideTests(fit.contr)
summary(dt)
```

```
##        CISvsWT
## Down      1662
## NotSig    8788
## Up        1519
```

Step 15

Display top 10 most differentially expressed genes:

```
options(digits=2)
topTable(fit.contr)
```

```
##        EntrezID             Symbol Length logFC AveExpr   t P.Value adj.P.Val
## 12700     12700               Cish   2161  -3.5   4.705 -27 2.1e-08   7.4e-05
## 69368     69368              Wdfy1   4714  -2.6   1.999 -21 1.6e-07   3.1e-04
## 21638     21638              Trgv4    419   3.8   3.666  17 5.9e-07   8.8e-04
## 436468   436468 Trav15d-1-dv6d-1    349   4.2   3.098  17 4.8e-07   8.1e-04
## 22290     22290                Uty   5228 -11.0  -2.841 -31 9.7e-09   7.4e-05
## 26908     26908            Eif2s3y   1767 -12.2  -0.086 -27 2.4e-08   7.4e-05
## 26900     26900              Ddx3y   4640 -11.5  -1.825 -27 2.5e-08   7.4e-05
## 20592     20592              Kdm5d   5471 -10.6  -2.515 -25 4.0e-08   9.6e-05
## 59310     59310              Myl10    880   2.6   3.140  14 2.1e-06   2.3e-03
## 13508     13508              Dscam   7481  -2.3   0.022 -14 2.3e-06   2.3e-03
##           B
## 12700  10.1
## 69368   8.2
## 21638   6.6
## 436468  6.3
## 22290   6.1
## 26908   6.0
## 26900   6.0
## 20592   5.8
## 59310   5.7
## 13508   5.6
```

Step 16

Save all differentially expressed genes to a file and all other results for further downstream analysis.

```
de.genes <- topTable(fit.contr, coef="CISvsWT", p.value=0.05, n=Inf)
write.csv(de.genes, file="DE_genes.csv", row.names=FALSE)
```

```
#save differential expression results
save(fit,fit.contr,dt,de.genes,y,file = "DE_results.RData")
```

Chapter 5

More details on the protocol

5.1 Sequencing quality

Sequencing output from a sequencer is stored in one or more FASTQ-format files. Each FASTQ file contains nucleotide sequences and sequencing quality strings for the reads generated from a library. The quality string for a read has the same length as the read sequence and each letter in the string encodes the sequencing quality score of the corresponding read base. The ASCII code of a letter is equal to the Phred quality score of the read base plus an integer offset. A Phred quality score denotes the sequencing quality of a read base. It is computed as -10 x log10P, where P is the probability that a read base is incorrectly called

The *qualityScores()* function in **Rsubread** extracts quality strings from a FASTQ file and returns the Phred scores in a data matrix object in which rows are reads and columns are base positions. Distribution of Phred scores at each base position can be viewed by using the *boxplot()* function. For reads generated by the popular Illumina sequencers such as HiSeq and NextSeq, sequencing quality is usually lower at the two ends of the read (particular at the 3' end) compared to the middle part of the read. The vast majority of read bases in a FASTQ file are expected to have a Phred score greater than 13 (corresponding to P value of 0.05). Fig. 2 shows the boxplots of quality scores in a library that was sequenced in 2015 by an Illumina HiSeq 2000 sequencer. The boxplots show that this dataset has a high sequencing quality.

Box 1 | The Simplified Annotation Format (SAF)

```
 The Simplified Annotation Format is defined in Rsubread and this format is used by
```

```
featureCounts to counts reads to genes or other genomic features. This format only
includes five compulsory columns: ‘GeneID’, ‘Chr’, ‘Start’, ‘End’ and
‘Strand’. Below is an example of SAF annotation for two RefSeq genes with Entrez
gene identifiers 497097 and 100503874.
GeneID    Chr  Start   End     Strand**
497097    chr1 3204563 3207049  -
497097    chr1 3411783 3411982  -
497097    chr1 3660633 3661579  -
100503874 chr1 3637390 3640590  -
100503874 chr1 3648928 3648985  -
Gene identifiers included in the column ‘GeneID’ can be integer numbers (eg. Entrez g
```

5.2 Gene annotation

The Rsubread package contains inbuilt gene annotation for human and mouse, making it convenient to process RNA-seq data generated from human or mouse samples. This annotation was generated based on the NCBI RefSeq gene annotation. In this annotation, overlapping exons from the same gene were merged to form a single exon covering all overlapping exons. The inbuilt annotations can be used in both read counting and read mapping. The inbuilt annotations are in SAF format (Box 1). They can be retrieved using the getInBuiltAnnotation() function in Rsubread.

Alternatively, external gene annotations such as those generated in Ensembl database (9) or GENCODE database (10) can be used for counting and mapping. Formats of external gene annotation that are accepted by Rsubread functions include GTF and GFF3 (https://genome.ucsc.edu/FAQ/FAQformat.html#format4).

5.3 Index building

An index built for a reference genome contains a large hash table, which stores chromosomal locations of subreads (16bp mers) extracted from the reference genome. With the hash table, subreads extracted from a read can be quickly mapped to the genome via a quick search (hashing) in the table. Candidate mapping locations of the read will then be determined by the voting of mapped subreads and final mapping location will be determined by further evaluation of candidate mapping locations.

The index can be built as either a full index or a gapped index. A full index contains subreads extracted from every base location of the genome, whereas a gapped index contains subreads extracted from every three bases in the genome. The use of a full index will make read mapping faster, but it also causes more computer memory to be used. Full index and gapped index can be further split into blocks. This can reduce memory usage but mapping time will increase on the other hand. See Box 2 for more information on gapped index and split index.

By default, *buildindex()* constructs a single-block full index for a reference genome. This allows maximum mapping speed to be achieved. To our knowledge, Rsubread is the only tool that allows users to tune the amount of memory used to achieve a desired balance between memory consumption and speed in read mapping. This flexibility allows Rsubread to be run on various computing platforms.

5.4 Read mapping

After an index is successfully built, sequencing reads can then be mapped to the reference genome via the *align()* function. Reads are mapped via a two-pass procedure. In the first pass, subreads (seeds) are extracted from each read and their locations in the genome are quickly determined by looking up the hash table included in the index. Mapped subreads then 'vote' for mapping locations of the read. This *'seed-and-vote'* strategy has been demonstrated to be more efficient and accurate than the conventional *'seed-and-extend'* strategy (Liao et al., 2019, 2013b). Indels will also be discovered in this step.

In the second pass, reads are re-aligned to the genome to determine their final mapping location by taking into account indels identified from the first pass. Note that *align()* does not perform full alignment for exon-spanning reads. It only aligns such reads to the exon they most overlap with.

Read bases overlapping other exons in such reads are soft-clipped. However this partial alignment is sufficient for RNA-seq expression quantification at the gene level because reads can be confidently assigned to one of the exons in the gene. To fully align exon-spanning reads, the *subjunc()* function in **Rsubread** can be used. *subjunc()* is slower than *align()*, but it is still computationally competitive. *subjunc()* is also a two-pass aligner. However it not only collects indels in its first pass, it also collects exon splicing sites. In its second pass, it will use detected exon splicing sites to further improve the mapping of exon-spanning reads.

```
Box 2 | Build an index with less memory usage

A gapped index can be built to reduce the index size.
Size of a gapped index is only one third of size of the full index.
Use of a gapped index could save more than 50% of memory used in read mapping.
During read mapping, the memory contains the index and also other data such as chromosomal sequer
For human or mouse genome, size of the full index is ~15GB and size of the gapped index is 5GB.
During read mapping, ~18GB of memory is consumed when full index is used and ~8GB when gapped inc
```

```
Below is the command for building a gapped index:
> buildindex("mm10_reference", "mm10.fa", gappedIndex=TRUE)
The full index and gapped index can be split into blocks to further
control the amount of memory to be used.
Users can specify the maximum memory allowed and  buildindex() will
automatically split the index into required number of blocks.
The index blocks will be loaded to memory sequentially during read mapping.
The more blocks an index is split into, the smaller the memory required for read mappi
The following command builds a full index that is split into multiple blocks to achiev
> buildindex("mm10_reference", "mm10.fa", indexSplit=TRUE,memory=4000)
```

Rsubread aligners are the first to use the two-pass strategy to improve the mapping quality of RNA-seq reads. This strategy has later been adopted by other RNA-seq aligners. Mapping results from *align()* include both mapped and unmapped reads. The results are saved to files in BAM or SAM format. Indels identified in the mapping are saved to VCF-format files. *align()* also returns an **R** object that contains mapping statistics for each library.

Percentage of successfully mapped reads in a library is affected by multiple factors including sequencing quality, read length, paired-end or single-end, quality of reference sequences and aligner setting. For the single-end reads generated from the latest Illumina HiSeq/NextSeq/NovaSeq sequencers, *align()* typically reports around 80-90 percent of mapped reads. If paired-end reads are provided, the mapping percentage is expected to be slightly higher.

5.5 Read counting

After read mapping is completed, mapped reads can be counted to genes by *featureCounts()* function in **Rsubread**. *featureCounts()* compares chromosomal coordinates of mapped reads with chromosomal coordinates of exons in each gene. *CIGAR* (Concise Idiosyncratic Gapped Alignment Report) string of each mapped read is analyzed so that read assignment can be done precisely. *featureCounts()* provides a wide range of counting options. For example, overlap between a read and an exon can be determined based on the number of overlapping bases, fraction of overlapping bases and number of non-overlapping bases. Multi-mapping reads can be excluded from counting, or only have their primary alignment counted or have all their alignments counted.

Similarly, for multi-overlapping reads that overlap more than one gene they can be excluded from counting or be counted to all their overlapping genes.

featureCounts() accepts both name-sorted and location-sorted reads. It can output counting details for each individual read.

featureCounts() returns an R object that includes a count table, gene annotation and counting statistics. Percentage of successfully assigned reads is affected by many factors including cell type, annotation quality and mRNA purity. The counting percentage could be highly variable between experiments, even for well annotated genomes such as human and mouse genomes. Typically around 50 – 90 percent of mapped reads are assigned to genes in a mouse experiment. For the RNA-seq data used in this protocol, ~70 percent of reads were successfully assigned.

5.6 Gene filtering

Genes that do not express or express at a very low level in all the samples are filtered out from analysis. Such genes are very hard to handle in the statistical testing. Typically ~50 percent of genes are excluded in the analysis of mouse RNA-seq data. For the RNA-seq data used in this protocol, 58 percent of genes were removed from analysis.

5.7 Sample clustering

Examining the clustering of samples is a useful way to check the quality of samples. Sample replicates should cluster together and different sample types should separate from each other.

5.8 Statistical testing of differential expression

A rigorous *false discovery rate (FDR)* threshold should be applied when calling differentially expressed genes. This threshold is usually *0.05*, but it might be relaxed to *0.1* or even *0.15* in some circumstances. The *treat()* function in **limma** might be considered for testing against a fold change of gene expression.

Chapter 6

Downstream analysis

In this chapter we will run through a series of example downstream analyses that can be performed on the identified differentially expressed genes.

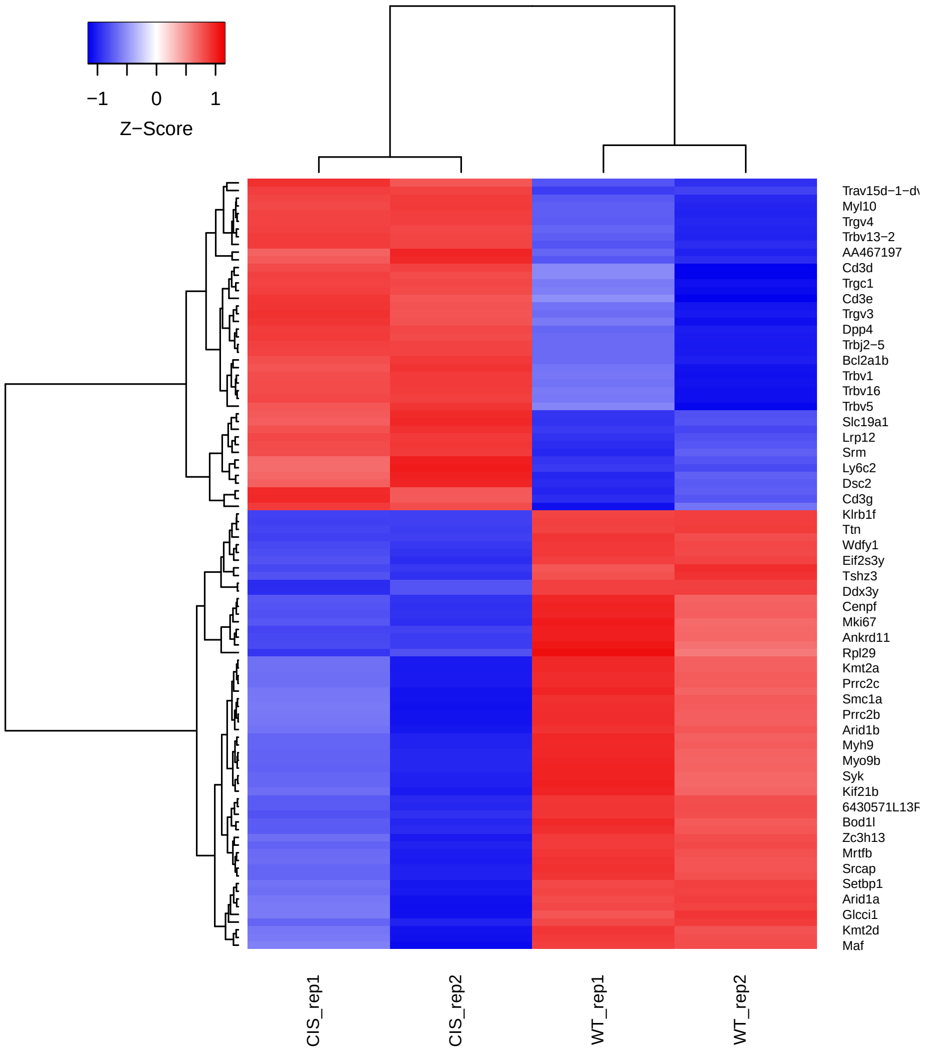
6.1 Visualisations

There are a number of ways in which you can visualise your RNA-seq data, while in this chapter we attempt to cover the most commonly ways of visualising your RNA-seq data, this is by no means exhaustive.

6.1.1 Heatmaps

Heatmaps are a great way of demonstrating the differences in the expression patterns between different conditions in your dataset. Here we use the *coolmap* function within the **limma** package which is an extension of the *heatmap.2* function within the **gplots** package. The heatmap below shows the expression pattern of the top 100 differentially expressed genes between CIS vs WT.

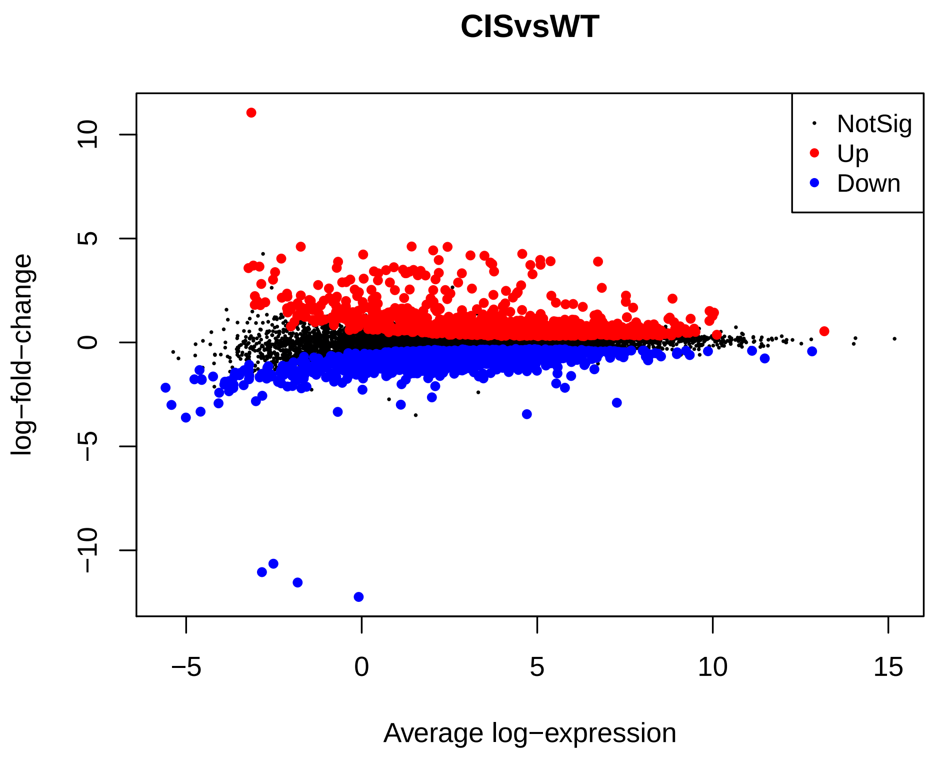
```
#load the required package
library(limma)
load("Counts.rdata")
load("DE_results.RData")
heat.data <- y$E[head(rownames(de.genes), n = 100), ]
colnames(heat.data) <- targets$Sample
rownames(heat.data) <- head(de.genes$Symbol, n = 100)
coolmap(
  heat.data,
  margin = c(5, 4),
  cexCol = 0.8,
  lhei = c(0.8, 4),
  cexRow = 0.6
)
```



6.1.2 Mean-difference plot

A mean-difference plot or MD-plot is a plot that can be used to show the fold-change differences against the average expression values of all genes used in the analysis. This can be generated easily using the *plotMD* function within the **limma** package.

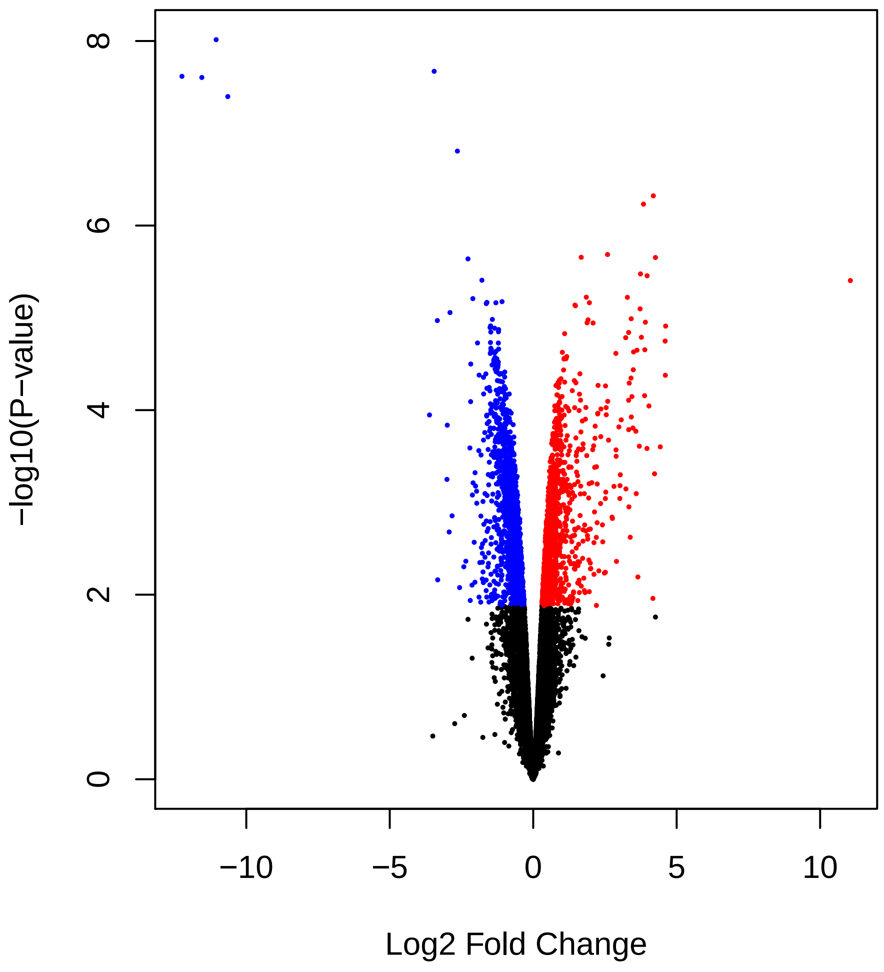
```r
plotMD(fit.contr, column = 1,status = dt,cex=0.8)
```



6.1.3 Volcano plot

A volcano plot shows the relationship between the log fold-change on the x-axis and a measure of statistical significance on the y-axis. The measure of significance can be -log(p-value) or the B-statistics. Here, we will use the *volcanoplot* function within the **limma** package.In the volcano plot below, differentially expressed genes are highlighted with red for up-regulated and blue for down-regulated genes.

```r
#highlight de genes
col<-c("blue","black","red")[factor(dt[,"CISvsWT"])]
volcanoplot(fit.contr,col=col)
```



6.2 Gene Ontology (GO) and KEGG enrichment analysis

6.2.1 Gene Ontology

Gene ontology (http://www.geneontology.org/) provides a controlled vocabulary for describing biological processes (BP ontology), molecular functions (MF ontology) and cellular components (CC ontology).

The GO ontologies themselves are organism-independent; terms are associated with genes for a specific organism through direct experimentation or through sequence homology with another organism and its GO annotation.

Terms are related to other terms through parent-child relationships in a directed acylic graph.

You can use enrichment analysis as another way of drawing conclusions from your set of differentially expressed genes.

```
enrich.pvalue<-0.00001
```

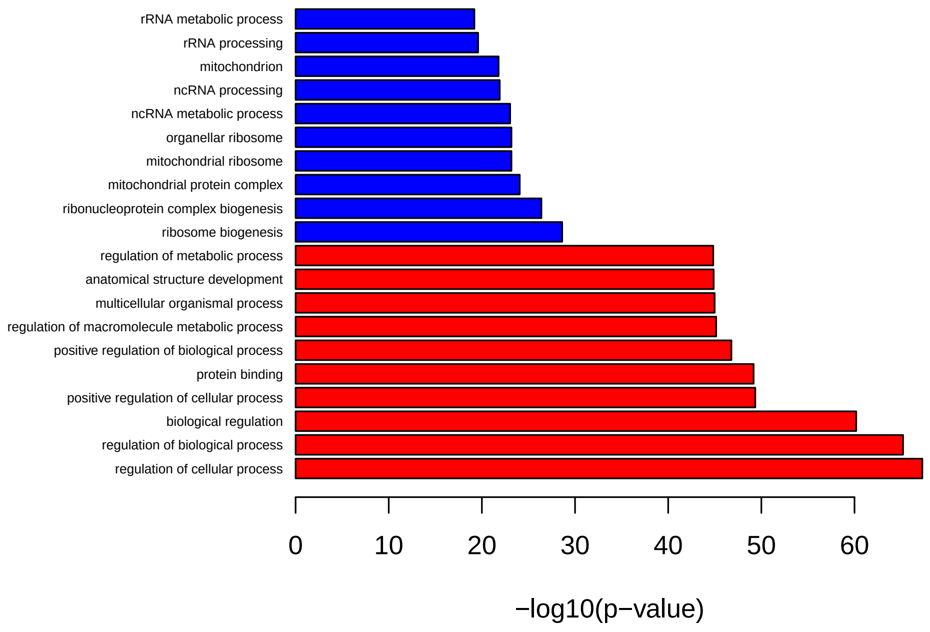
Here, we use the *goana* function within the **limma** package to test for enrichment of differentially expressed genes between the CIS vs WT. Note that the *p-values* returned by *goana* are **unadjusted for multiple testing**. It is therfore, advisable that if the results are to be published, only terms with very small p-values should be included. For instance, in the example below, only terms with a p-value $< 10^{-5}$ are retained.

```
if(!requireNamespace("GO.db"))
  BiocManager::install("GO.db")
library(GO.db)
go.rst<-goana(fit.contr,coef = 1,FDR = 0.05,species="Mm")
#order enriched terms by p-value
go.rst<-go.rst[with(go.rst,P.Up<enrich.pvalue|P.Down<enrich.pvalue),]
go.rst<-topGO(go.rst,number = Inf)
topGO(go.rst)
```

```
##                                                              Term Ont    N  Up
## GO:0050794                           regulation of cellular process  BP 6084 593
## GO:0050789                         regulation of biological process  BP 6343 628
## GO:0065007                                    biological regulation  BP 6647 675
## GO:0048522                positive regulation of cellular process  BP 3439 335
## GO:0005515                                          protein binding  MF 5634 584
## GO:0048518              positive regulation of biological process  BP 3697 368
## GO:0060255           regulation of macromolecule metabolic process  BP 3917 361
## GO:0032501                         multicellular organismal process  BP 3570 335
## GO:0048856                         anatomical structure development  BP 3061 287
## GO:0019222                          regulation of metabolic process  BP 4221 404
## GO:0005488                                                  binding  MF 7993 876
## GO:0032502                                    developmental process  BP 3334 319
## GO:0031323                 regulation of cellular metabolic process  BP 3857 361
## GO:0048731                                       system development  BP 2484 228
## GO:0007275                       multicellular organism development  BP 2804 261
## GO:0016043                          cellular component organization  BP 3824 409
## GO:0006996                                   organelle organization  BP 2732 266
## GO:0010604 positive regulation of macromolecule metabolic process  BP 2214 224
## GO:0051171      regulation of nitrogen compound metabolic process  BP 3590 321
## GO:0080090               regulation of primary metabolic process  BP 3689 332
##            Down P.Up  P.Down
## GO:0050794 1177    1 5.2e-68
## GO:0050789 1206    1 6.1e-66
## GO:0065007 1233    1 6.5e-61
## GO:0048522  746    1 4.5e-50
## GO:0005515 1069    1 6.7e-50
## GO:0048518  779    1 1.6e-47
## GO:0060255  808    1 7.0e-46
## GO:0032501  754    1 1.0e-45
## GO:0048856  673    1 1.3e-45
## GO:0019222  853    1 1.4e-45
## GO:0005488 1360    1 1.5e-45
## GO:0032502  716    1 2.1e-45
## GO:0031323  796    1 6.2e-45
## GO:0048731  575    1 1.4e-44
## GO:0007275  628    1 1.7e-44
## GO:0016043  789    1 2.7e-44
## GO:0006996  607    1 2.5e-41
## GO:0010604  518    1 1.4e-40
## GO:0051171  738    1 1.8e-39
## GO:0080090  753    1 2.2e-39
```

A simple way to visualise the enrichment results is through a bar plot as shown below which shows the top 10 enriched terms

```
top.up <- head(go.rst[order(go.rst$P.Up, decreasing = F), ], n = 10)
top.down <- head(go.rst[order(go.rst$P.Down, decreasing = F), ], n = 10)
bar.data <-
  rbind(
    data.frame(
      Term = top.up$Term,
      P.value = top.up$P.Up,
      Dir = "up"
    ),
    data.frame(
      Term = top.down$Term,
      P.value = top.down$P.Down,
      Dir = "down"
    )
  )
bar.data <- bar.data[order(bar.data$P.value, decreasing = F), ]
par(mai = c(0.8, 2, 0.5, 0.5))
bb<-barplot(
  -log10(bar.data$P.value),
  horiz = T,
  xlab = " -log10(p-value)",
  cex.names = 0.5,
  col = c("red","blue")[factor(bar.data$Dir)]
)
axis(2,line = -0.8,at=bb,labels = bar.data$Term,tick = F,las=2,cex.axis=0.5)
```



Add gene information (optional)

Sometimes you may want to know what genes are enriched in each of the terms, this section allows you to add gene details.

```
library(org.Mm.eg.db)
go.EntrezID <- as.list(org.Mm.egGO2ALLEGS)
go.rst <- tibble::rownames_to_column(go.rst, var = "GO.ID")

#Add gene details
go.rst$Genes.Up <- unlist(lapply(go.rst$GO.ID, function(x) {
  xx <- go.EntrezID[[x]]
  if (is.null(xx) |
      with(go.rst[go.rst$GO.ID == x,], Up == 0 |
           P.Up > enrich.pvalue))
    return("-")
  x <- rownames(dt[rownames(dt) %in% xx & dt == 1,])
  x <-
    paste0(with(fit.contr$genes, Symbol[EntrezID %in% xx]), collapse = "|")
  return(x)
}))

go.rst$Genes.Down <- unlist(lapply(go.rst$GO.ID, function(x) {
  xx <- go.EntrezID[[x]]
  if (is.null(xx) |
      with(go.rst[go.rst$GO.ID == x,], Down == 0 |
           P.Down > enrich.pvalue))
    return("-")
  x <- rownames(dt[rownames(dt) %in% xx & dt == -1,])
  x <-
    paste0(with(fit.contr$genes, Symbol[EntrezID %in% xx]), collapse = "|")
  return(x)
}))
```

Bibliography

Delconte, R. B., Kolesnik, T. B., Dagley, L. F., Rautela, J., Shi, W., Putz, E. M., Stannard, K., Zhang, J.-G., Teh, C., Firth, M., et al. (2016). Cis is a potent checkpoint in nk cell–mediated tumor immunity. *Nature Immunology*, 17(7):816–824.

Law, C. W., Chen, Y., Shi, W., and Smyth, G. K. (2014). voom: Precision weights unlock linear model analysis tools for rna-seq read counts. *Genome biology*, 15(2):R29.

Liao, Y., Smyth, G. K., and Shi, W. (2013a). featurecounts: an efficient general purpose program for assigning sequence reads to genomic features. *Bioinformatics*, 30(7):923–930.

Liao, Y., Smyth, G. K., and Shi, W. (2013b). The subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic acids research*, 41(10):e108–e108.

Liao, Y., Smyth, G. K., and Shi, W. (2019). The R package Rsubread is easier, faster, cheaper and better for alignment and quantification of RNA sequencing reads. *Nucleic Acids Research*, 47:e47.

O'Leary, N. A., Wright, M. W., Brister, J. R., Ciufo, S., Haddad, D., McVeigh, R., Rajput, B., Robbertse, B., Smith-White, B., Ako-Adjei, D., et al. (2015). Reference sequence (refseq) database at ncbi: current status, taxonomic expansion, and functional annotation. *Nucleic acids research*, 44(D1):D733–D745.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ritchie, M. E., Phipson, B., Wu, D., Hu, Y., Law, C. W., Shi, W., and Smyth, G. K. (2015). limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research*, 43(7):e47.

Shi, W., Liao, Y., and with contributions from Jenny Dai, G. K. S. (2020). *Rsubread: Mapping, quantification and variant analysis of sequencing data*. R package version 2.4.2.

Smyth, G., Hu, Y., Ritchie, M., Silver, J., Wettenhall, J., McCarthy, D., Wu, D., Shi, W., Phipson, B., Lun, A., Thorne, N., Oshlack, A., de Graaf, C., Chen, Y., Langaas, M., Ferkingstad, E., Davy, M., Pepin, F., and Choi, D. (2020). *limma: Linear Models for Microarray Data.* R package version 3.46.0.

Smyth, G. K. (2004). Linear models and empirical bayes methods for assessing differential expression in microarray experiments. *Statistical applications in genetics and molecular biology*, 3(1).