Lab 2: Quality Control, Normalization, Differential Expression, and Enrichment

Omics Data Science

Himel Mallick

Contents

Setup

```
library(SummarizedExperiment)
library(airway)
```

```
library(DESeq2)
library(edgeR)
library(ComplexHeatmap)
library(ggplot2)
library(EDASeq)
library(topGO)
library(tidyverse)
library(clusterProfiler)
library(gsEasy)
library(GOfuncR)
theme_set(theme_bw())
```

The Airway data

```
data(airway)
airway
#> class: RangedSummarizedExperiment
#> dim: 63677 8
#> metadata(1): ''
#> assays(1): counts
#> rownames(63677): ENSG00000000003 ENSG00000000005 ... ENSG00000273492
#>   ENSG00000273493
#> rowData names(10): gene_id gene_name ... seq_coord_system symbol
#> colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
#> colData names(9): SampleName cell ... Sample BioSample
colData(airway)
#> DataFrame with 8 rows and 9 columns
#>             SampleName     cell      dex    albut        Run avgLength
#>               <factor> <factor> <factor> <factor>   <factor> <integer>
#> SRR1039508 GSM1275862  N61311     untrt    untrt SRR1039508       126
#> SRR1039509 GSM1275863  N61311     trt      untrt SRR1039509       126
#> SRR1039512 GSM1275866  N052611    untrt    untrt SRR1039512       126
#> SRR1039513 GSM1275867  N052611    trt      untrt SRR1039513        87
#> SRR1039516 GSM1275870  N080611    untrt    untrt SRR1039516       120
#> SRR1039517 GSM1275871  N080611    trt      untrt SRR1039517       126
#> SRR1039520 GSM1275874  N061011    untrt    untrt SRR1039520       101
#> SRR1039521 GSM1275875  N061011    trt      untrt SRR1039521        98
#>             Experiment    Sample    BioSample
#>               <factor>  <factor>     <factor>
#> SRR1039508  SRX384345 SRS508568 SAMN02422669
#> SRR1039509  SRX384346 SRS508567 SAMN02422675
#> SRR1039512  SRX384349 SRS508571 SAMN02422678
#> SRR1039513  SRX384350 SRS508572 SAMN02422670
#> SRR1039516  SRX384353 SRS508575 SAMN02422682
#> SRR1039517  SRX384354 SRS508576 SAMN02422673
#> SRR1039520  SRX384357 SRS508579 SAMN02422683
#> SRR1039521  SRX384358 SRS508580 SAMN02422677
```

Quality Control - Gene filtering

In this dataset, we have measurements for 63677 genes. Not all of them will be expressed in this particular system.

```
table(rowSums(assay(airway))>0)
#>
#> FALSE  TRUE
#> 30208 33469
```

In general, it is advisable to remove those genes with a low expression level. Often, we may consider a filter that keeps those genes that are expressed in at least some reads in a certain number of samples.

The `filterByExpr` function of the `edgeR` package does that, considering by default 10 reads in at least 10 samples.

```
filter <- filterByExpr(airway)
table(filter)
#> filter
#> FALSE  TRUE
#> 49453 14224
filtered <- airway[filter,]
filtered
#> class: RangedSummarizedExperiment
#> dim: 14224 8
#> metadata(1): ''
#> assays(1): counts
#> rownames(14224): ENSG00000000003 ENSG00000000419 ... ENSG00000273382
#>   ENSG00000273486
#> rowData names(10): gene_id gene_name ... seq_coord_system symbol
#> colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
#> colData names(9): SampleName cell ... Sample BioSample
```

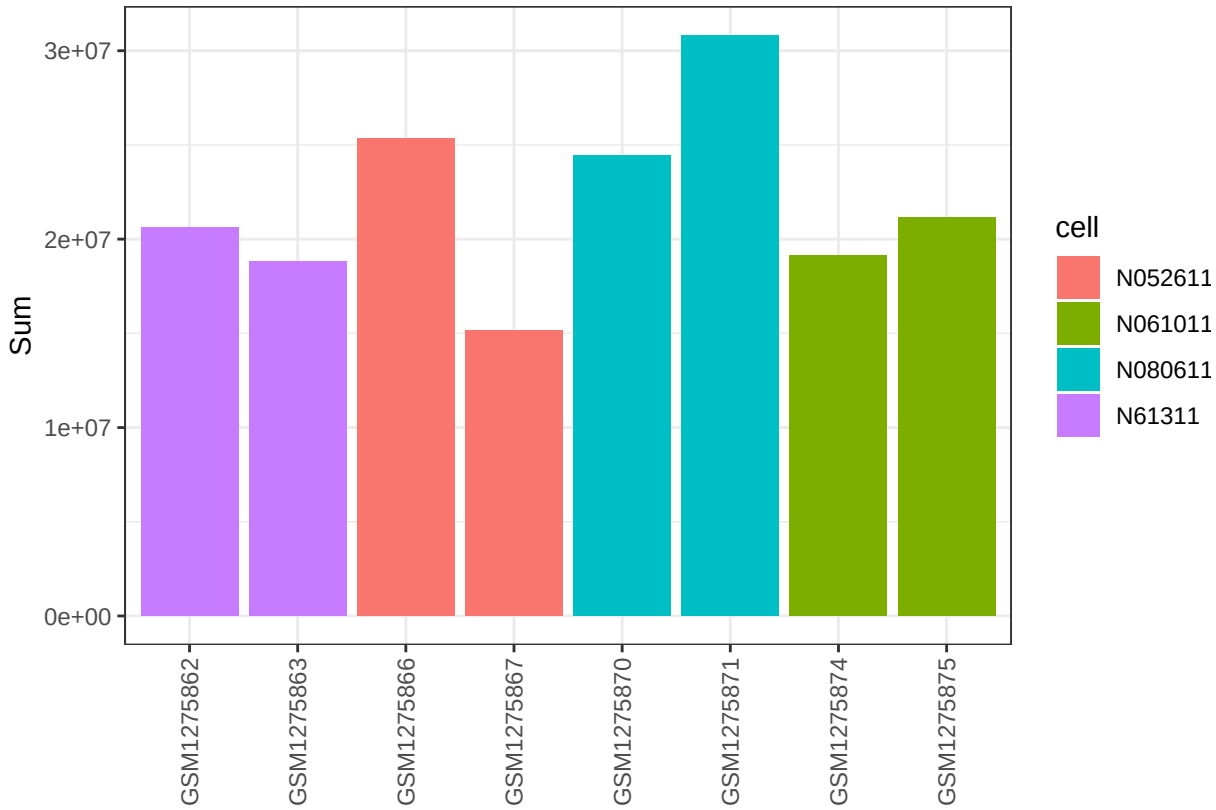
Quality Control - Exploratory Data Analysis

Exploratory Data Analysis (EDA) is useful to highlight those genes that have lower quality or that can have a high leverage in the downstream analyses.

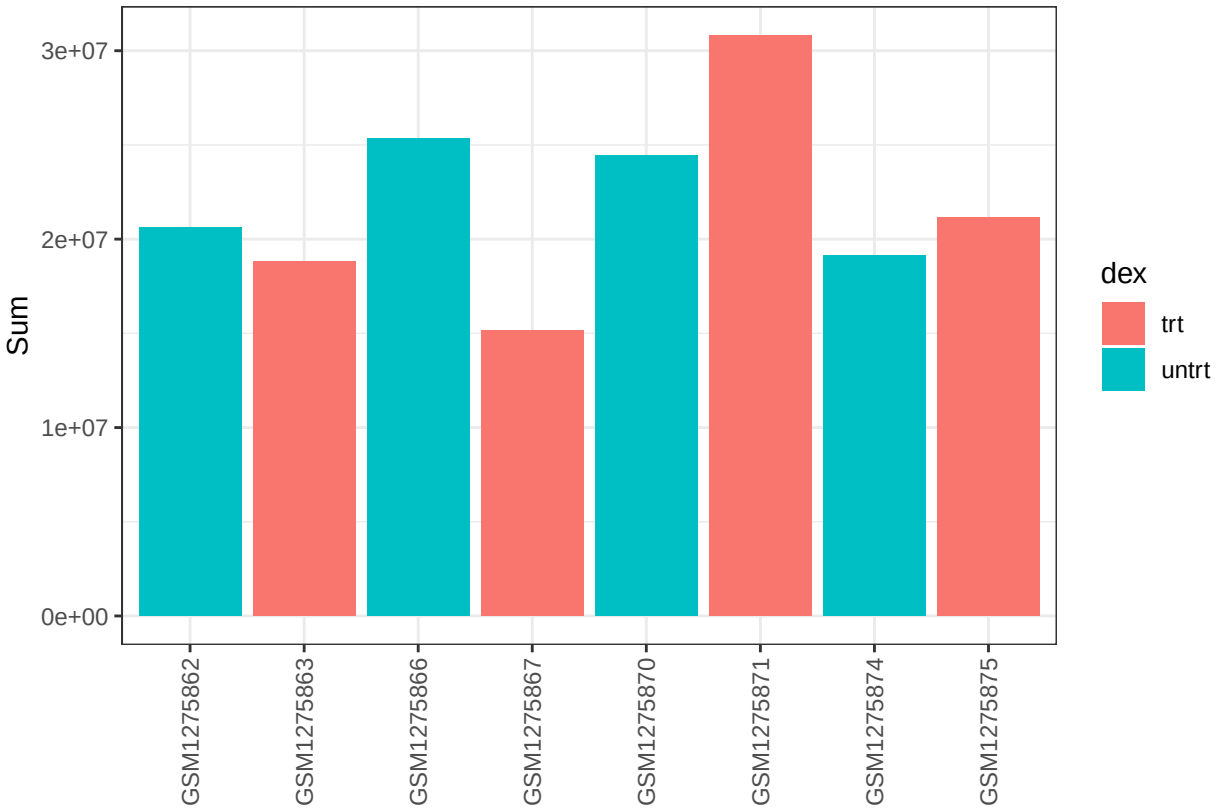
Total number of reads

First, we need to compute the total number of mapped reads per sample.

```
airway$Sum <- colSums(assay(airway))

df <- as.data.frame(colData(airway))

ggplot(df, aes(x=SampleName, y=Sum, fill=cell)) +
   geom_bar(stat="identity") + theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  xlab('')
```




```
ggplot(df, aes(x=SampleName, y=Sum, fill=dex)) +
  geom_bar(stat="identity") + theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  xlab('')
```



While there is clearly a difference in the total number of reads, it does not appear to be related to either cell type or treatment.

Remember that the `SummarizedExperiment` object may contain multiple matrices (with the same dimension). Hence, we can "save" the log-transformed matrix in the object to avoid computing the log every time that we want to plot their distribution.

```
assay(filtered, "logcounts") <- log1p(assay(filtered))
filtered
#> class: RangedSummarizedExperiment
#> dim: 14224 8
#> metadata(1): ''
#> assays(2): counts logcounts
#> rownames(14224): ENSG00000000003 ENSG00000000419 ... ENSG00000273382
#>   ENSG00000273486
#> rowData names(10): gene_id gene_name ... seq_coord_system symbol
#> colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
#> colData names(9): SampleName cell ... Sample BioSample
```

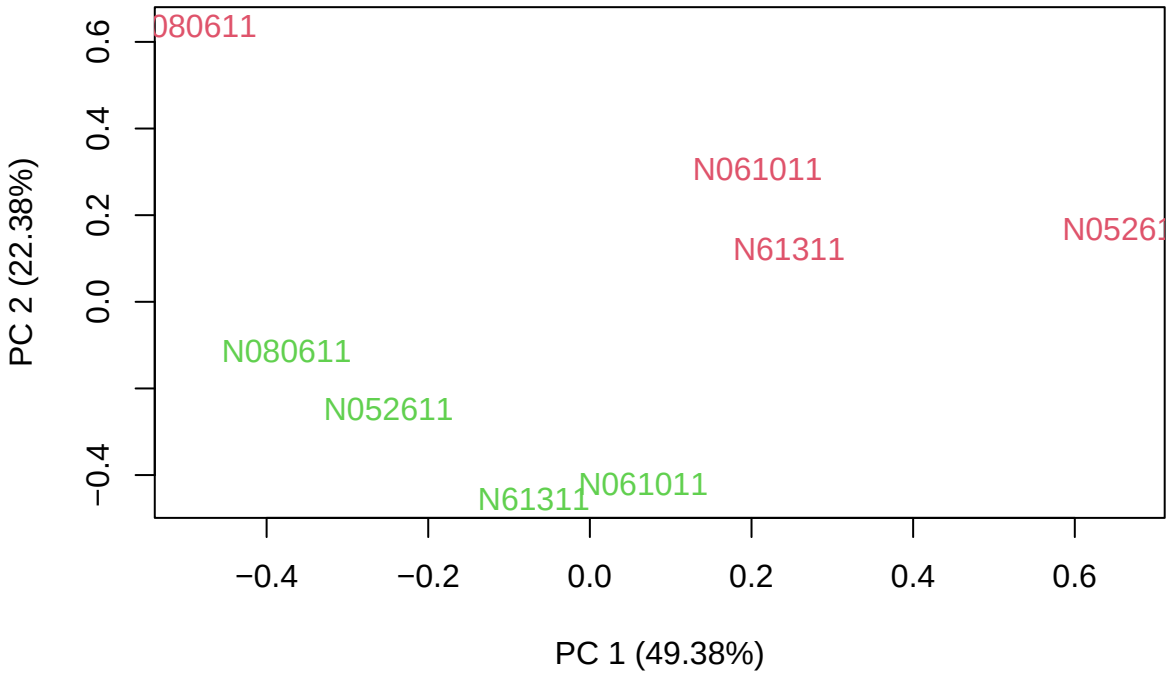
Principal Component Analysis (PCA)

To understand which sample characteristics influence gene expression, it is a good idea to observe the first two or three principal components. Ideally, we would like the samples to group by biological variables and not technical noise.

The `plotPCA` function of the `EDASeq` package can be used for this goal. Alternatively, one can use any of the R packages to compute the principal components.

```
tmp <- assay(filtered)
colnames(tmp) <- filtered$cell
```

```
pal <- palette()[-1]
plotPCA(tmp, col=pal[filtered$dex])
```



Normalization

Most of the differences observed in the above box plots are not attributable to biological signals but rather to technical variations between samples, such as the total number of reads sequenced for each sample.

We can attempt to mitigate these differences through a normalization procedure. Various normalization methods have been proposed for RNA-seq data, but the simplest and often most effective is to scale each sample by a constant.

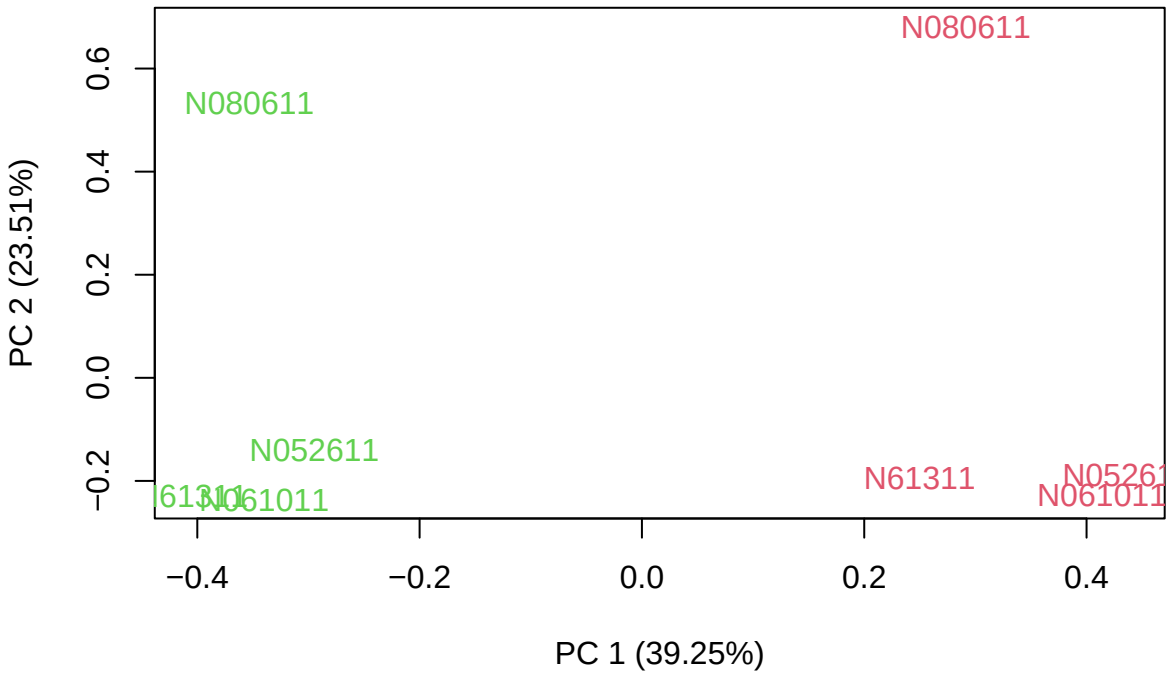
The `EDASeq` package implements several such methods, including the 'upper-quartile' method, which aligns the upper quartile of the distribution of each sample.

Again, we can store the normalized matrix into an assay of the object.

```
assay(filtered, "uq") <- betweenLaneNormalization(assay(filtered), which="upper")
```

A useful diagnostic plot to verify that the normalized data look good is the plot of the first two principal components. We expect the direction of most variability to be explained by biology rather than technical differences.

```
tmp <- assay(filtered, "uq")
colnames(tmp) <- filtered$cell
plotPCA(tmp, col=pal[filtered$dex])
```



Unlike unnormalized data, after normalization the first principal component separates treated and control samples.

Count-based differential expression analysis

Now we can create a `DESEqDataSet` object: this is the object that we will use to conduct our differential expression analysis.

Note that we start from the `SummarizedExperiment` object and we add a *design matrix* that specifies which columns of the `colData` we will use as covariates in the statistical model.

```
dds <- DESeqDataSet(filtered,
                    design = ~ cell + dex)
dds
#> class: DESeqDataSet
#> dim: 14224 8
#> metadata(2): '' version
#> assays(3): counts logcounts uq
#> rownames(14224): ENSG00000000003 ENSG00000000419 ... ENSG00000273382
#>   ENSG00000273486
#> rowData names(10): gene_id gene_name ... seq_coord_system symbol
#> colnames(8): SRR1039508 SRR1039509 ... SRR1039520 SRR1039521
#> colData names(9): SampleName cell ... Sample BioSample
```

Note that the `DESeqDataSet` class "inherits" from `SummarizedExperiment`, meaning that an object of this class is also a `SummarizedExperiment` and all its methods are still applicable.

```
class(dds)
#> [1] "DESeqDataSet"
#> attr(,"package")
#> [1] "DESeq2"
is(dds, "SummarizedExperiment")
#> [1] TRUE
```

The `DESeq2` package (as well as `edgeR` the other popular package for differential expression) uses a strategy that models the RNA-seq read counts directly using a Negative Binomial Generalized Linear Model (GLM).

Hence, unlike other approaches, we will apply our statistical model directly to the count data, without the need for data transformation.

In particular, for each gene, we fit a GLM with the formula specified above and we test the null hypothesis

$$H_0 : \beta = 0$$

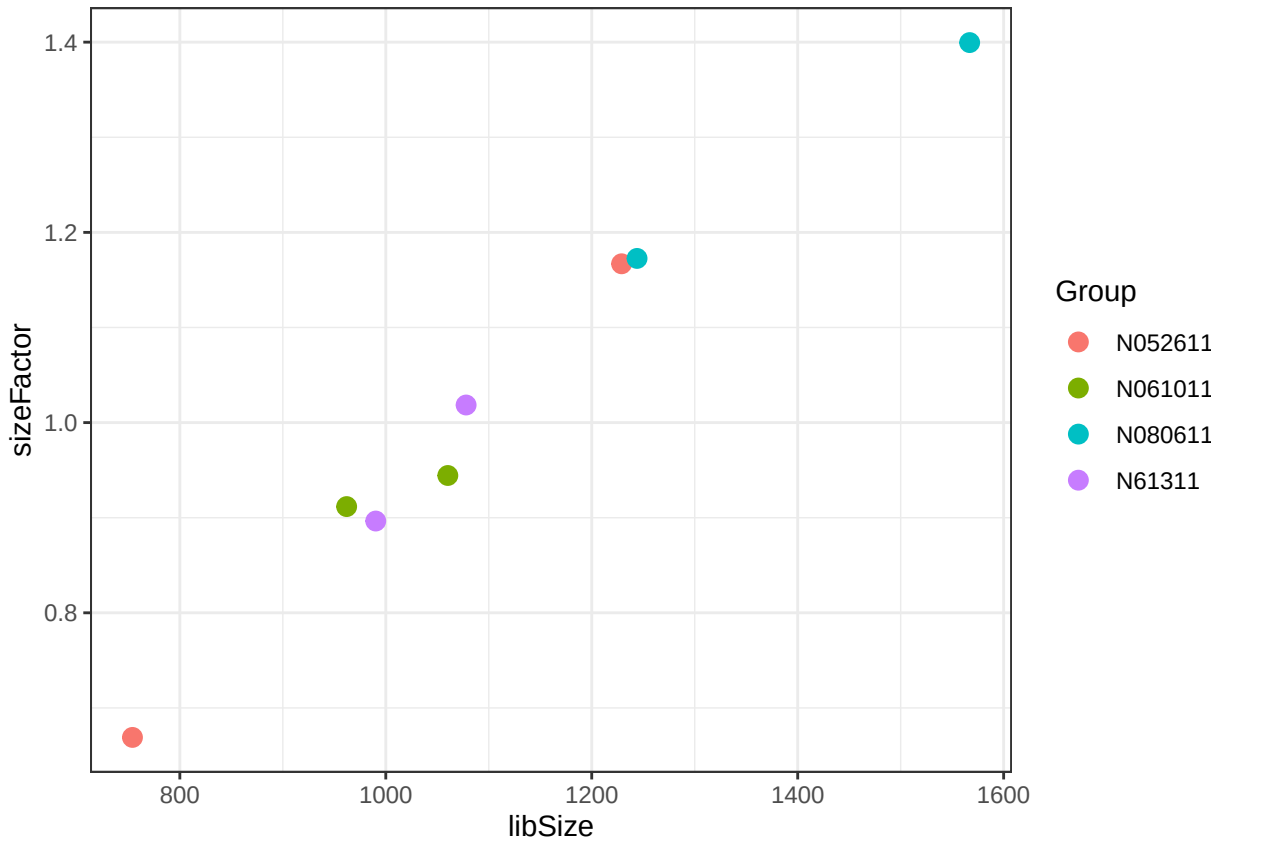
against a bilateral alternative, where β is the coefficient of the covariate that we want to test (to be specified later), in this case `dex`, the treatment.

Normalization in the context of differential expression analysis

What we discussed about normalization is still relevant: we do not want to misinterpret technical differences as biological ones and identify genes that appear to differ between treated and control samples only because of technical reason.

However, transforming the data via normalization is not an option since we want our GLM to model count data. The solution is to estimate the *size factors* and include them in the model as *offsets*.

```
dds <- estimateSizeFactors(dds)

ggplot(data.frame(libSize = apply(assay(dds), 2, quantile, .75),
                  sizeFactor = sizeFactors(dds),
                  Group = dds$cell),
       aes(x = libSize, y = sizeFactor, col = Group)) +
    geom_point(size = 3)
```



This scatter plot shows that there is a positive correlation which makes sense because scaling factors essentially convert the library sizes into `effective` library sizes .

Dispersion Estimation

Usually, the Poisson distribution is a good starting point for count data. However, RNA-seq data are *overdispersed*, meaning that the variance is greater than the mean.

For this reason, DESeq2 and edgeR use a negative binomial that assumes that the variance is a quadratic function of the mean:

$$V(\mu) = \mu + \phi\mu^2,$$

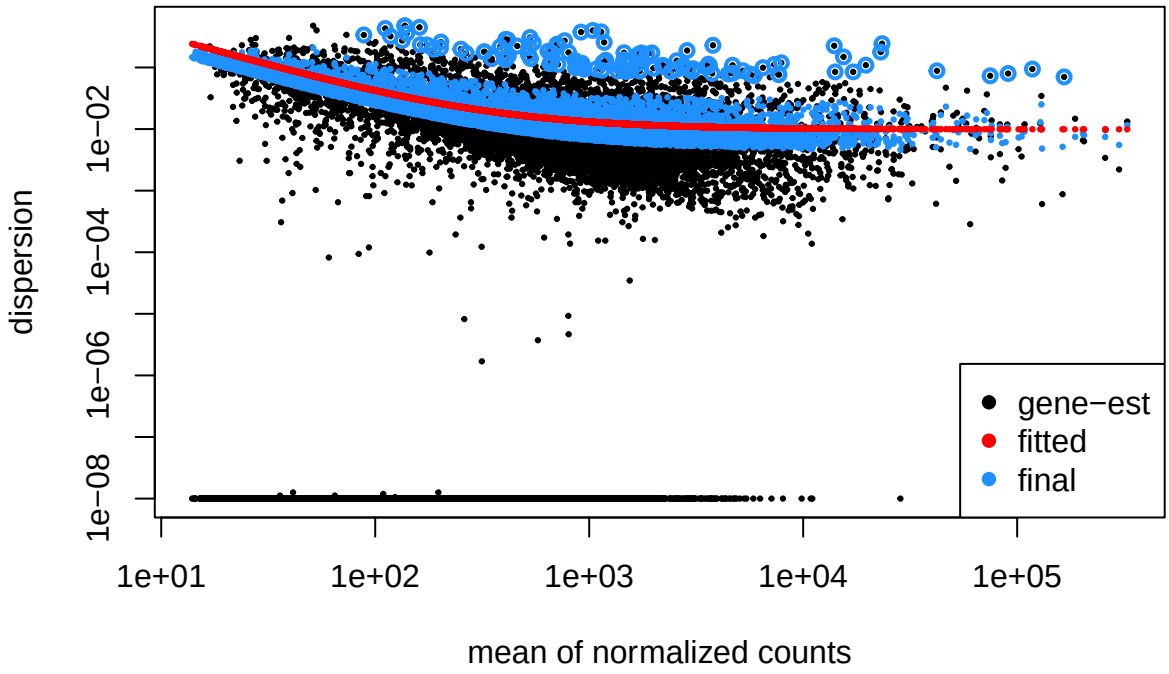
where ϕ is the *dispersion parameter*, to be estimated for each gene.

Since we typically have very large p (genes) and small-ish n (samples), a good strategy is to "borrow strength" across genes to improve the accuracy in the estimation of the dispersion parameters.

DESeq2's strategy is to first compute a preliminary genewise estimate via maximum likelihood estimation, then estimate the mean-dispersion relationship fitting a regression line and finally *shrinking* the estimates towards the fit. All three steps are performed within the `estimateDispersions` function.

```
dds <- estimateDispersions(dds)
```

```
plotDispEsts(dds)
```



Test for DE

Finally, we can use a Wald test to test the differential expression between, say, treated and control.

```
dds <- nbinomWaldTest(dds)
```

Note that the wrapper function `DESeq()` performs these three steps automatically.

Explore the results

```
res <- results(dds, contrast = c("dex", "trt", "untrt"))
res
#> log2 fold change (MLE): dex trt vs untrt
#> Wald test p-value: dex trt vs untrt
#> DataFrame with 14224 rows and 6 columns
#>                   baseMean log2FoldChange     lfcSE       stat      pvalue
#>                  <numeric>      <numeric> <numeric>  <numeric>   <numeric>
#> ENSG00000000003  711.6507     -0.3916468 0.1002736  -3.905781 9.39217e-05
#> ENSG00000000419  522.1894      0.1964258 0.1115943   1.760178 7.83776e-02
#> ENSG00000000457  238.0809      0.0275546 0.1412405   0.195090 8.45322e-01
#> ENSG00000000460   58.1603     -0.0998310 0.2799299  -0.356628 7.21370e-01
#> ENSG00000000971 5837.3125      0.4159432 0.0891274   4.666837 3.05872e-06
#> ...                    ...            ...       ...        ...         ...
#> ENSG00000273329   43.7610     -0.2277364  0.292112 -0.7796191    0.435615
#> ENSG00000273344   66.2977      0.0316459  0.273984  0.1155026    0.908047
#> ENSG00000273373   25.9784     -0.0362499  0.363926 -0.0996078    0.920656
#> ENSG00000273382   18.2479     -0.7414291  0.470279 -1.5765744    0.114893
#> ENSG00000273486   15.5160     -0.1595433  0.473073 -0.3372490    0.735929
#>                       padj
#>                  <numeric>
#> ENSG00000000003 6.94718e-04
#> ENSG00000000419 1.94495e-01
#> ENSG00000000457 9.20965e-01
#> ENSG00000000460 8.48067e-01
#> ENSG00000000971 3.15971e-05
#> ...                     ...
#> ENSG00000273329    0.633006
#> ENSG00000273344    0.953245
#> ENSG00000273373    0.959511
#> ENSG00000273382    0.259486
#> ENSG00000273486    0.855986
```

```
summary(res)
#>
#> out of 14224 with nonzero total read count
#> adjusted p-value < 0.1
#> LFC > 0 (up)       : 2441, 17%
#> LFC < 0 (down)     : 2194, 15%
#> outliers [1]       : 0, 0%
#> low counts [2]     : 0, 0%
#> (mean count < 14)
#> [1] see 'cooksCutoff' argument of ?results
#> [2] see 'independentFiltering' argument of ?results
```

```
head(res[order(res$pvalue), ])
#> log2 fold change (MLE): dex trt vs untrt
#> Wald test p-value: dex trt vs untrt
#> DataFrame with 6 rows and 6 columns
#>                   baseMean log2FoldChange     lfcSE      stat       pvalue
#>                  <numeric>      <numeric> <numeric> <numeric>    <numeric>
#> ENSG00000152583   997.958        4.56442  0.184002   24.8063 7.66744e-136
#> ENSG00000165995   495.667        3.28040  0.132306   24.7940 1.04119e-135
```

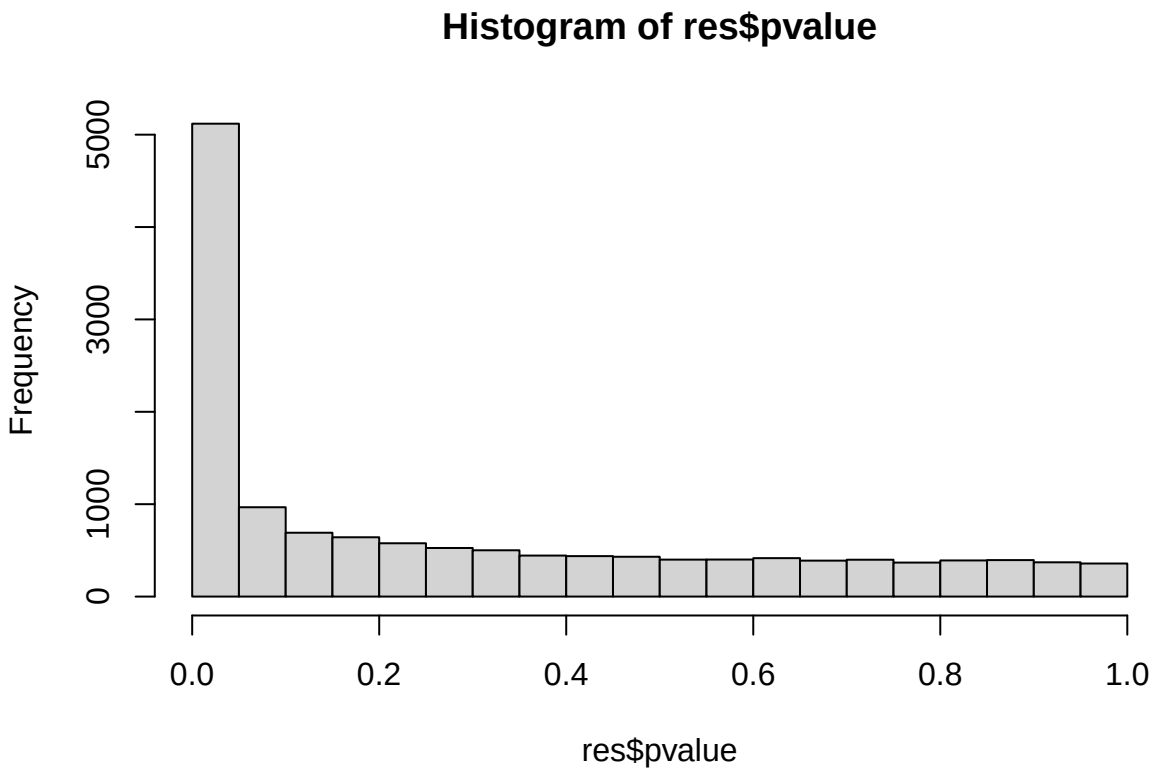
```
#> ENSG00000101347 12714.667       3.75653  0.155699   24.1268 1.30789e-128
#> ENSG00000120129  3413.120       2.93737  0.121942   24.0882 3.32352e-128
#> ENSG00000189221  2343.870       3.34320  0.141829   23.5721 7.45395e-123
#> ENSG00000211445 12298.969       3.71989  0.166701   22.3147 2.65795e-110
#>                         padj
#>                    <numeric>
#> ENSG00000152583 7.40494e-132
#> ENSG00000165995 7.40494e-132
#> ENSG00000101347 6.20115e-125
#> ENSG00000120129 1.18184e-124
#> ENSG00000189221 2.12050e-119
#> ENSG00000211445 6.30111e-107
```

Since we have fitted a model for each gene, we need to take into account the multiple testing problem and consequently adjust the p-values. DESeq2 uses the Benjamini-Hochberg procedure to control the false discovery rate (FDR).

Histogram of p-values

One good diagnostic plot to check that the model fit is good is to look at the distribution of the p-values. We expect them to be a mixture of a uniform distribution (for those genes that are not DE) and a spike at 0 (for the DE genes).

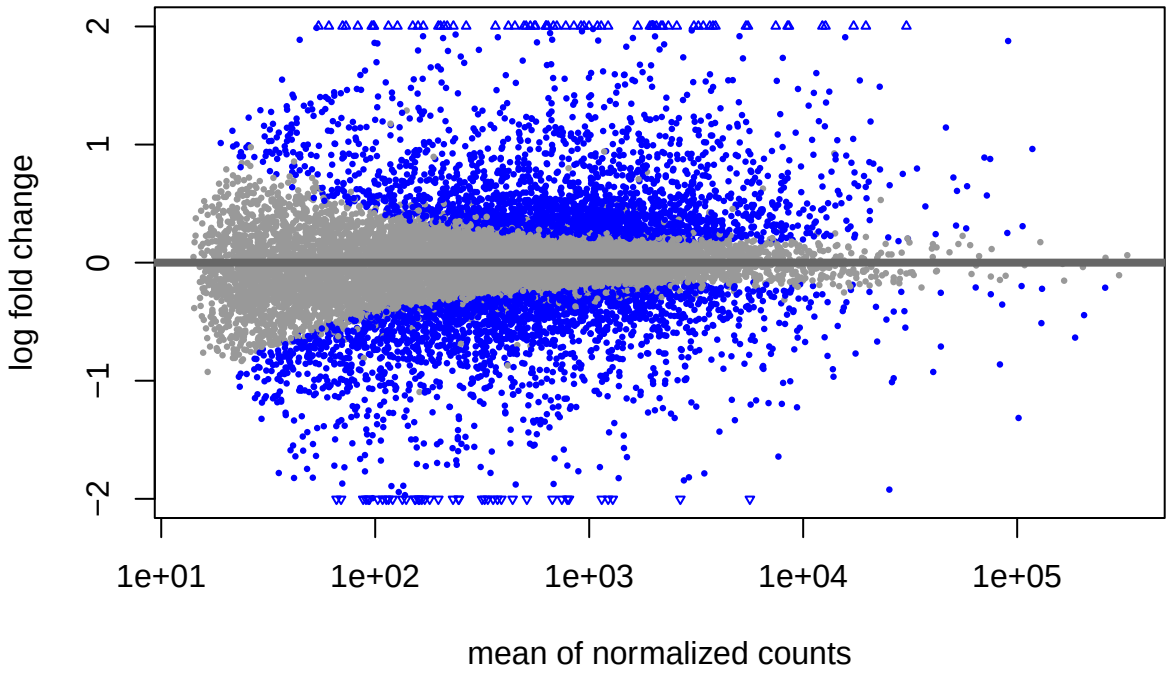
```
hist(res$pvalue)
```



MA-plot and volcano plot

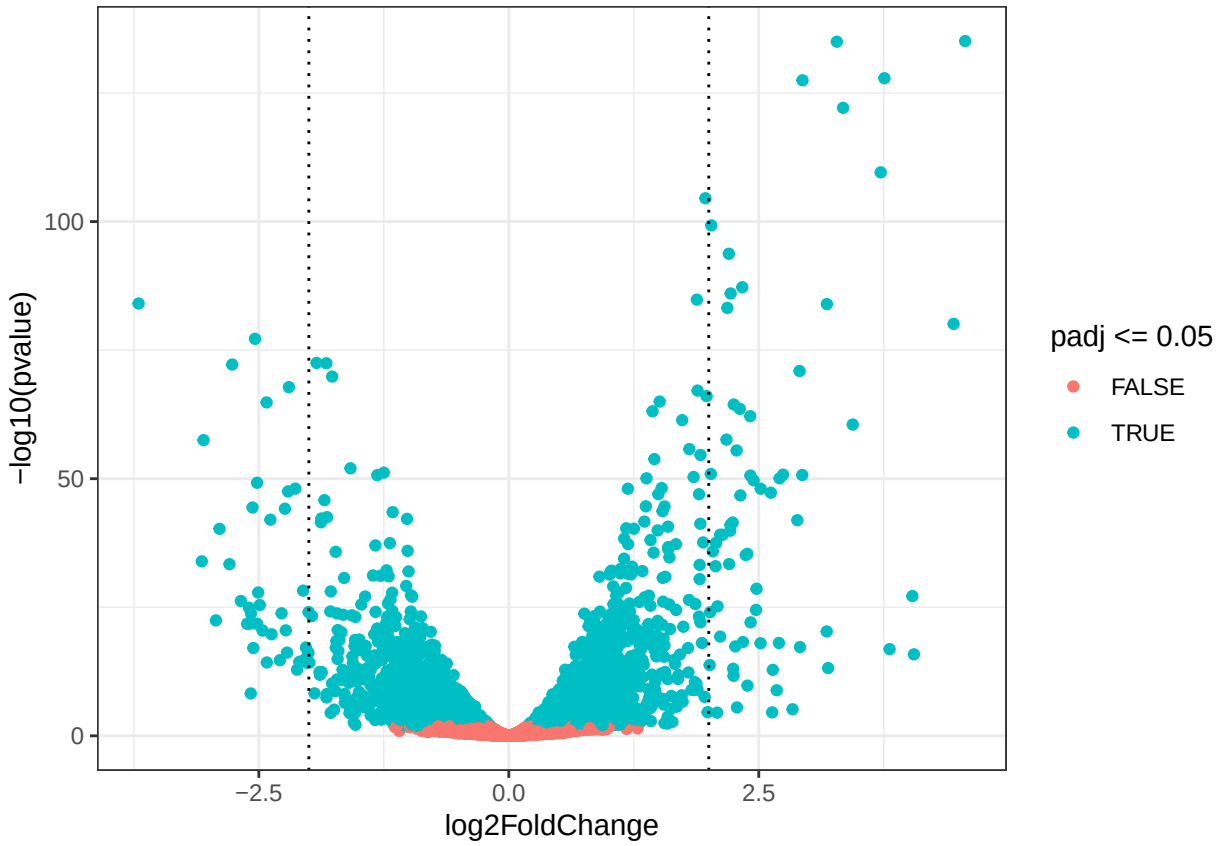
A good way to visualize differential expression is the MA-plot (aka Mean-Difference plot). It consists of a 45-degree rotation of the scatterplot between the treated and control mean expression.

```
DESeq2::plotMA(res)
```



Finally, the *volcano plot* is a way to simultaneously look at the significance (on the y axis) and effect size (x axis) of differential expression.

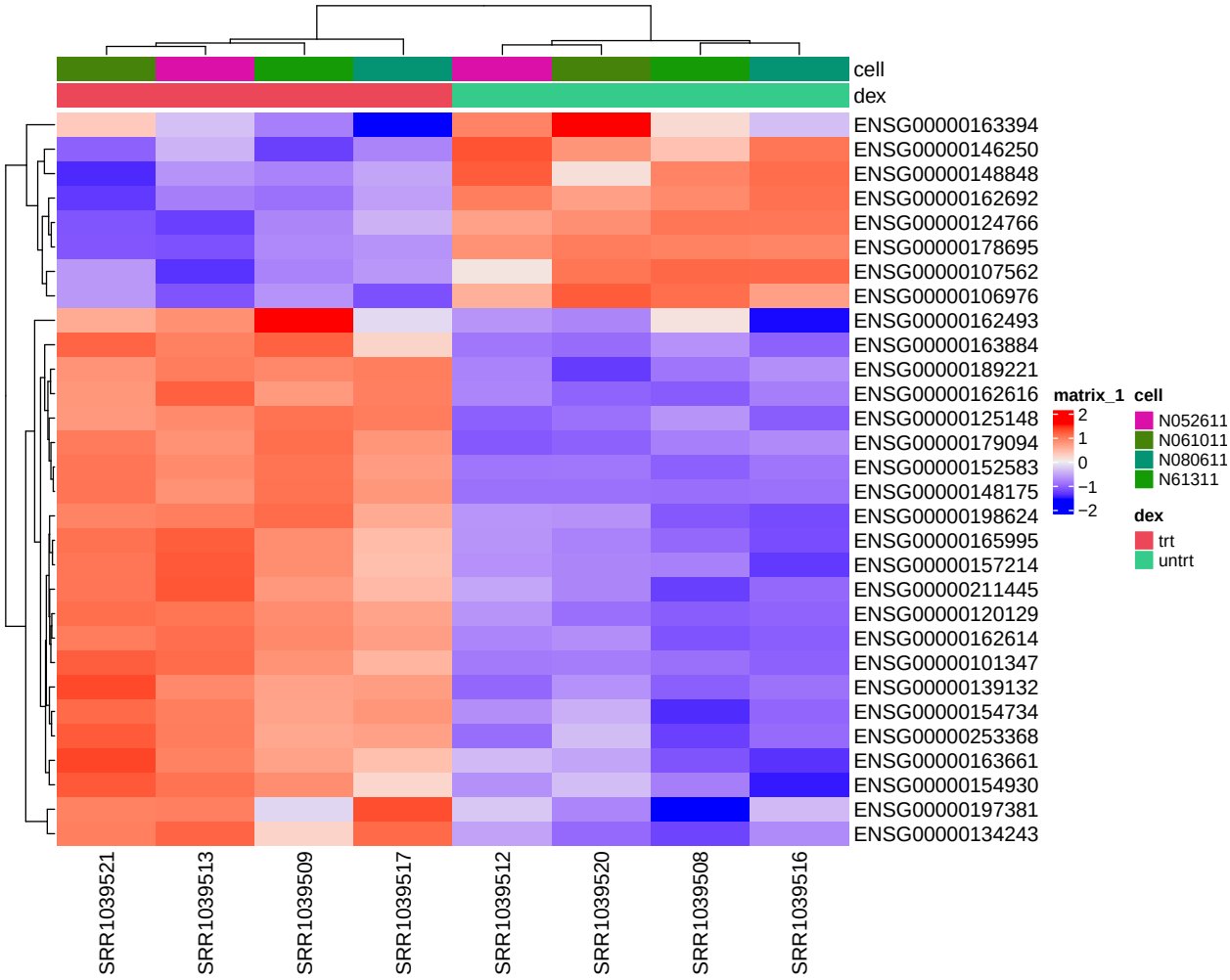
```
ggplot(as.data.frame(res),
       aes(x = log2FoldChange, y = -log10(pvalue), color = padj<=0.05)) +
    geom_point() +
    geom_vline(xintercept = c(-2, 2), linetype = "dotted")
```



Visualize selected genes

Very often we are interested in looking at the most differentially expressed genes ("top hits"). In this case, we focus on the top 30 genes ordered by p-value and visualize their expression in a heatmap.

```
vsd <- vst(dds, blind = TRUE)

# Get top DE genes
genes <- res[order(res$pvalue), ] |>
         head(30) |>
         rownames()
heatmapData <- assay(vsd)[genes, ]

# Scale counts for visualization
heatmapData <- t(scale(t(heatmapData)))

# Add annotation
heatmapColAnnot <- data.frame(colData(vsd)[, c("cell", "dex")])
heatmapColAnnot <- HeatmapAnnotation(df = heatmapColAnnot)


# Plot as heatmap
ComplexHeatmap::Heatmap(heatmapData,
                        top_annotation = heatmapColAnnot,
                        cluster_rows = TRUE, cluster_columns = TRUE)
```



The heatmap shows a clear separation in the gene expression of these genes between the treated and control groups, as expected.

Gene Set Enrichment Analysis (GSEA)

Going from genes to gene sets

So far we have seen how to use omics data to derive a list of statistically significant differentially expressed (DE) genes, while controlling for false discovery. Sometimes it can be comvenient to look at biological pathways, or more generally gene sets to gain actionable biological insights.

Goals of GSEA

Detecting changes in gene expression datasets can be hard due to - the large number of genes/features, - the high variability between samples, and - the limited number of samples.

The goal of GSEA is to enable the detection of modest but coordinated changes in prespecified sets of related genes. Such a set might include all the genes in a specific pathway, for instance, or genes that have been shown to be coregulated based on previously published studies.

GSEA aggregates the per gene statistics across genes within a gene set, therefore making it possible to detect situations where all genes in a predefined set change in a small but coordinated way. This corresponds to the

hypothesis that many relevant phenotypic differences are manifested by small but consistent changes in a set of genes/features.

Early GSEA methods - Over Representation Analysis

Over Representation Analysis (ORA) is a widely used approach to determine whether known biological functions or processes are over-represented (= enriched) in an experimentally-derived gene list, *e.g.* a list of DE genes (DEGs).

The p-value can be calculated by the hypergeometric distribution.

$$p = 1 - \sum_{i=0}^{k-1} \frac{\binom{M}{i}\binom{N-M}{n-i}}{\binom{N}{n}}$$

In this equation, `N` is the total number of genes in the background distribution, `M` is the number of genes within that distribution that are annotated (either directly or indirectly) to the gene set of interest, `n` is the size of the list of genes of interest and `k` is the number of genes within that list which are annotated to the gene set. The background distribution by default is all the genes that have annotation. P-values should be adjusted for multiple comparison.

Example: Gene expression study, in which 11,812 genes have been tested for differential expression between two sample conditions and 202 genes were found DE.

Among the DE genes, 25 are annotated to a specific functional gene set, which contains in total 262 genes. This setup corresponds to a 2x2 contingency table:

```
d <-  matrix(c(25, 237, 177, 11323),
          nrow = 2,
          dimnames = list(c("DE", "Not DE"),
                          c("In GS", "Not in GS")))
d
#>         In GS Not in GS
#> DE         25       177
#> Not DE    237     11323
```

whether the overlap of 25 genes are significantly over represented in the gene set can be assessed using hypergeometric distribution. This corresponding to a one-sided version of Fisher's exact test.

```
fisher.test(d, alternative = "greater")
#>
#>  Fisher's Exact Test for Count Data
#>
#> data:  d
#> p-value = 2.815e-12
#> alternative hypothesis: true odds ratio is greater than 1
#> 95 percent confidence interval:
#>  4.502824      Inf
#> sample estimates:
#> odds ratio
#>   6.744738
```

For algorithm details, please refer to the vignette of DOSE.

Modern GSEA using the gsEasy package

A major limitation of ORA is that it restricts analysis to DE genes, excluding the vast majority of genes not satisfying the chosen significance threshold.

This is resolved by modern GSEA. Given a priori defined set of gene S (e.g., genes sharing the same *GO* category), the goal of GSEA is to determine whether the members of S are randomly distributed throughout the ranked gene list (L) or primarily found at the top or bottom.

There are two key elements of the GSEA method:

- Calculation of an Enrichment Score.
  - The enrichment score (ES) represent the degree to which a set S is over-represented at the top or bottom of the ranked list L. The score is calculated by walking down the list L, increasing a running-sum statistic when we encounter a gene in S and decreasing when it is not. The magnitude of the increment depends on the gene statistics (e.g., correlation of the gene with phenotype). The ES is the maximum deviation from zero encountered in the random walk; it corresponds to a weighted Kolmogorov-Smirnov-like statistic.
- Esimation of Significance Level of ES.
  - The p-value of the ES is calculated using permutation test. Specifically, we permute the gene labels of the gene list L and recompute the ES of the gene set for the permutated data, which generate a null distribution for the ES. The p-value of the observed ES is then calculated relative to this null distribution.

We will utilize the `gsEasy` R package, which serves as a comprehensive tool for performing gene set enrichment analysis based on arbitrary gene sets, provided that a mapping between a group of gene sets and their members is available. This sets it apart from other packages, which usually only cater to a predefined set of signatures.

Input data for GSEA

For ORA, all we need is a gene vector, that is a vector of gene IDs. These gene IDs can be obtained by differential expression analysis (*e.g.* with DESeq2 package).

For GSEA, we need a ranked list of genes. `gsEasy` calculates p-values of enrichment for sets (of genes) in ranked/scored lists (of genes) by permutation. Please check out the associated vignette for more details.

GSEA as applied to the DE genes in the Airway Data

We first load the gene sets.

```
########################################
# Load DE results and rank by p-values #
########################################

results<-as.data.frame(res[order(res$pvalue, decreasing = FALSE),])

####################################
# Load mapping and create gene sets #
####################################

mapping = bitr(rownames(results), fromType = "ENSEMBL", toType = "GO", OrgDb = "org.Hs.eg.db")
mapping_info<- get_names(mapping$GO)
colnames(mapping_info)<-c('ID', 'Name', 'Process')
mod.gs = tapply(mapping$GO, mapping$ENSEMBL, as.character)
geneSet<-inverseList(mod.gs)
```

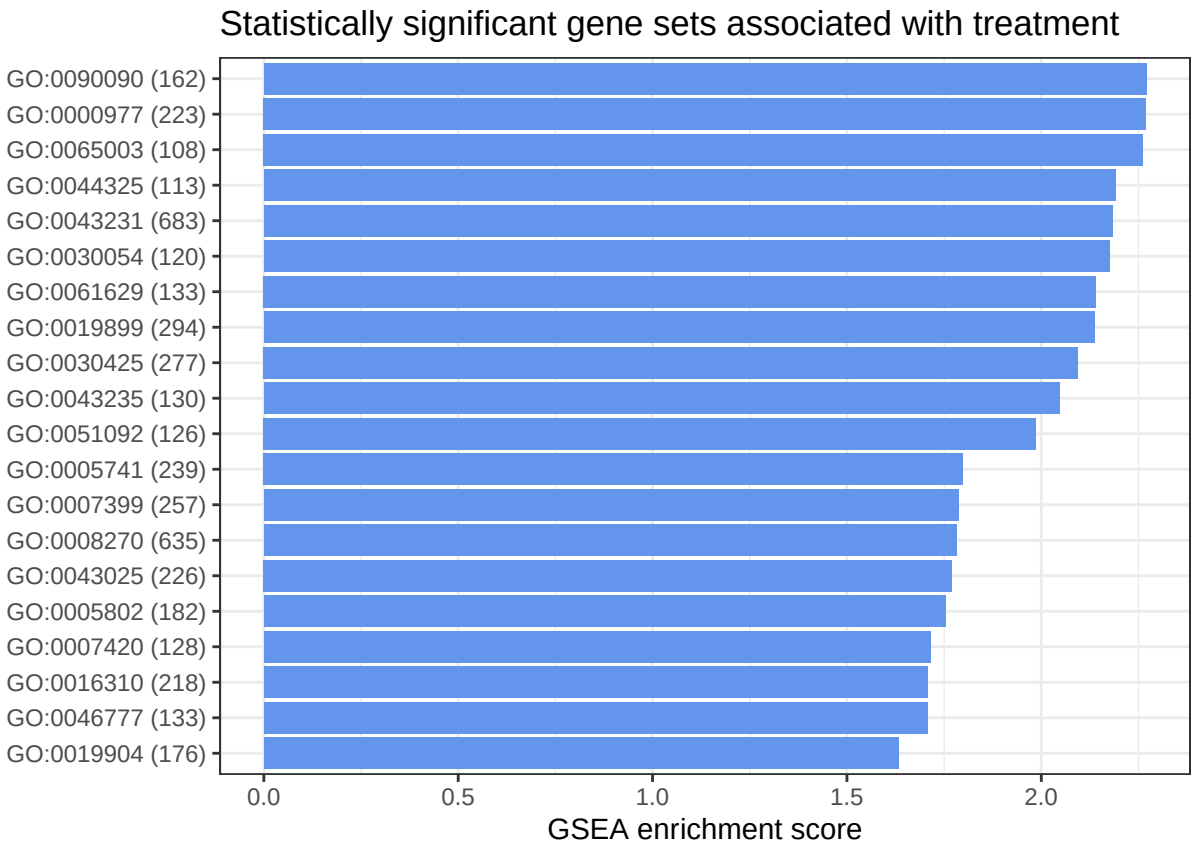
The wrapper below does the following steps: it filters out gene sets with a very small number of genes (<100), runs 1000 permutations per gene set using `gsEasy` (default is one million), and returns the enrichment acores after FDR correction. **NOTE: THIS STEP IS COMPUTATIONALLY INTENSIVE!**

```
################
# GSEA Wrapper #
################

run_GSEA<-function(geneSet, # A list of gene sets
                   ranked_features, # Ranked list of featured
                   filter.count = 100,
                   seed = 1234,
                   min_its = 200,
                   max_its = 1000,
                   fdr.correction = 'BH') {


  ####################
  # Filter out sets #
  ####################

  geneSet0<-geneSet
  cond <- sapply(geneSet0, function(x) length(x) > filter.count)
  geneSet<-geneSet0[cond]
  lengthGeneSet<-as.data.frame(reshape2::melt(lapply(geneSet, function(x) length(x))))
  colnames(lengthGeneSet)<-c('Freq','Set')

  #################
  # Classic GSEA #
  #################

  set.seed(seed)
  enrichment<-as.data.frame(sapply(geneSet, function(set) gset(S=set, r=ranked_features)))
  colnames(enrichment)<-'ES'
  enrichment<-rownames_to_column(enrichment, 'Set')
  enrichment<-merge(enrichment, lengthGeneSet, 'Set')
  enrichment$qval<-p.adjust(enrichment$ES, fdr.correction)

  ##########
  # Return #
  ##########

  return(enrichment)

}

############
# Run GSEA #
############

GSEA<-run_GSEA(geneSet, rownames(results))
GSEA<-dplyr::select(GSEA, c('Set', 'Freq', 'ES'), everything())
colnames(GSEA)<-c('ID', 'Size', 'pval', 'qval')


####################
# Merge with names #
```

```
####################

mapping_info_subset<-mapping_info %>% filter(ID %in% GSEA$ID) %>% filter(!duplicated(.))
GSEA<-merge(GSEA, mapping_info_subset, 'ID')
head(GSEA[, 1:5])
#>           ID Size         pval         qval
#> 1 GO:0000122  813 0.0001999980 6.849932e-04
#> 2 GO:0000139  608 0.0000099999 5.708276e-05
#> 3 GO:0000151  120 0.7014925373 8.284869e-01
#> 4 GO:0000209  167 0.6119402985 7.485341e-01
#> 5 GO:0000226  116 0.0325496745 6.557802e-02
#> 6 GO:0000278  106 0.7313432836 8.491019e-01
#>                                                          Name
#> 1 negative regulation of transcription by RNA polymerase II
#> 2                                                Golgi membrane
#> 3                                     ubiquitin ligase complex
#> 4                                   protein polyubiquitination
#> 5                         microtubule cytoskeleton organization
#> 6                                              mitotic cell cycle
```

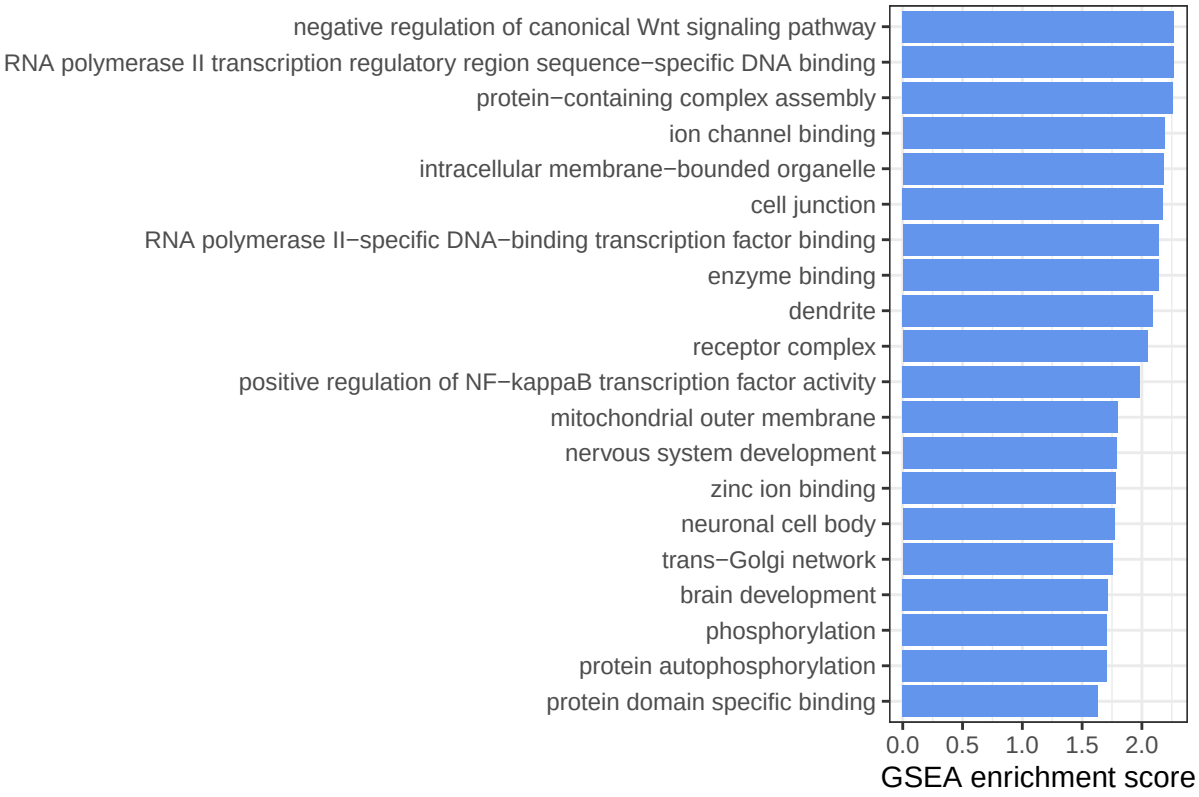
We can plot the top 20 gene sets that are statistically significant after FDR correction.

```
########
# Plot #
########

p<-GSEA %>%
  filter(qval < 0.05) %>%
  top_n(n = 20, wt = abs(qval)) %>%
  arrange(-pval) %>%
  mutate(ID = paste(ID, ' (', Size, ')', sep = ''))  %>%
  mutate(ID = factor(ID, levels = ID)) %>%
  ggplot(aes(y = -log10(pval), x = ID)) +
  geom_bar(stat = "identity", fill = 'cornflowerblue') + theme_bw() +
  coord_flip() +
  ggtitle('Statistically significant gene sets associated with treatment') +
  xlab('') +
  ylab('GSEA enrichment score')
print(p)
```



We can also plot a more meaningful representation of the top GO terms.

```
########
# Plot #
########

p<-GSEA %>%
  filter(qval < 0.05) %>%
  top_n(n = 20, wt = abs(qval)) %>%
  arrange(-pval) %>%
  mutate(Name = factor(Name, levels = Name)) %>%
  ggplot(aes(y = -log10(pval), x = Name)) +
  geom_bar(stat = "identity", fill = 'cornflowerblue') + theme_bw() +
  coord_flip() +
  ggtitle('') +
  xlab('') +
  ylab('GSEA enrichment score')
print(p)
```



Homework 2

- As we discussed in the class, the other popular package for differential expression analysis is `edgeR`. Re-run the analysis of this lab using the `edgeR` workflow. You can have a look at the associated vignette to understand which functions to use. What normalization method does `edgeR` use? How is the dispersion parameter estimated? Which test statistics did you use?
- Compare the results of the `edgeR` and `DESeq2` analyses. Which genes are found DE in both analyses? Which method finds the most DE genes? Explore the similarities and differences graphically.
- Repeat the GSEA analysis on the DE genes found by `edgeR`. How many gene sets are found enriched in both analyses? Explore the similarities and differences graphically.

Session Info

```
sessionInfo()
#> R version 4.4.2 (2024-10-31)
#> Platform: aarch64-apple-darwin20
#> Running under: macOS Sequoia 15.2
#>
#> Matrix products: default
#> BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
#> LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  LAPACK v
#>
#> locale:
#> [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
#>
```

```
#> time zone: Asia/Kolkata
#> tzcode source: internal
#>
#> attached base packages:
#> [1] grid      stats4    stats     graphics  grDevices utils     datasets
#> [8] methods   base
#>
#> other attached packages:
#>  [1] GOfuncR_1.24.0              vioplot_0.5.0
#>  [3] zoo_1.8-12                  sm_2.2-6.0
#>  [5] gsEasy_1.5                  clusterProfiler_4.12.6
#>  [7] lubridate_1.9.4             forcats_1.0.0
#>  [9] stringr_1.5.1               dplyr_1.1.4
#> [11] purrr_1.0.2                 readr_2.1.5
#> [13] tidyr_1.3.1                 tibble_3.2.1
#> [15] tidyverse_2.0.0             topGO_2.56.0
#> [17] SparseM_1.84-2              GO.db_3.19.1
#> [19] AnnotationDbi_1.66.0        graph_1.82.0
#> [21] EDASeq_2.38.0               ShortRead_1.62.0
#> [23] GenomicAlignments_1.40.0    Rsamtools_2.20.0
#> [25] Biostrings_2.72.1           XVector_0.44.0
#> [27] BiocParallel_1.38.0         ggplot2_3.5.1
#> [29] ComplexHeatmap_2.20.0       edgeR_4.2.2
#> [31] limma_3.60.6                DESeq2_1.44.0
#> [33] airway_1.24.0               SummarizedExperiment_1.34.0
#> [35] Biobase_2.64.0              GenomicRanges_1.56.2
#> [37] GenomeInfoDb_1.40.1         IRanges_2.38.1
#> [39] S4Vectors_0.42.1            BiocGenerics_0.50.0
#> [41] MatrixGenerics_1.16.0       matrixStats_1.4.1
#>
#> loaded via a namespace (and not attached):
#>   [1] splines_4.4.2           BiocIO_1.14.0           ggplotify_0.1.2
#>   [4] bitops_1.0-9            filelock_1.0.3          R.oo_1.27.0
#>   [7] polyclip_1.10-7         XML_3.99-0.17           lifecycle_1.0.4
#>  [10] httr2_1.0.7             pwalign_1.0.0           doParallel_1.0.17
#>  [13] lattice_0.22-6          MASS_7.3-61             magrittr_2.0.3
#>  [16] rmarkdown_2.29          yaml_2.3.10             cowplot_1.1.3
#>  [19] DBI_1.2.3               RColorBrewer_1.1-3      abind_1.4-8
#>  [22] zlibbioc_1.50.0         R.utils_2.12.3          ggraph_2.2.1
#>  [25] RCurl_1.98-1.16         yulab.utils_0.1.8       tweenr_2.0.3
#>  [28] rappdirs_0.3.3          circlize_0.4.16         GenomeInfoDbData_1.2.12
#>  [31] enrichplot_1.24.4       ggrepel_0.9.6           tidytree_0.4.6
#>  [34] codetools_0.2-20        DelayedArray_0.30.1     DOSE_3.30.5
#>  [37] xml2_1.3.6              ggforce_0.4.2           tidyselect_1.2.1
#>  [40] shape_1.4.6.1           aplot_0.2.4             UCSC.utils_1.0.0
#>  [43] farver_2.1.2            viridis_0.6.5           BiocFileCache_2.12.0
#>  [46] jsonlite_1.8.9          GetoptLong_1.0.5        tidygraph_1.3.1
#>  [49] iterators_1.0.14        foreach_1.5.2           tools_4.4.2
#>  [52] progress_1.2.3          treeio_1.28.0           Rcpp_1.0.13-1
#>  [55] glue_1.8.0              gridExtra_2.3           SparseArray_1.4.8
#>  [58] xfun_0.49               qvalue_2.36.0           withr_3.0.2
#>  [61] fastmap_1.2.0           latticeExtra_0.6-30     digest_0.6.37
#>  [64] gridGraphics_0.5-1      timechange_0.3.0        R6_2.5.1
```

```
#>  [67] colorspace_2.1-1        gtools_3.9.5            jpeg_0.1-10
#>  [70] biomaRt_2.60.1          RSQLite_2.3.9           R.methodsS3_1.8.2
#>  [73] generics_0.1.3          data.table_1.16.4       rtracklayer_1.64.0
#>  [76] prettyunits_1.2.0       graphlayouts_1.2.1      httr_1.4.7
#>  [79] S4Arrays_1.4.1          ontologyIndex_2.12      scatterpie_0.2.4
#>  [82] pkgconfig_2.0.3         gtable_0.3.6            blob_1.2.4
#>  [85] hwriter_1.3.2.1         shadowtext_0.1.4        htmltools_0.5.8.1
#>  [88] fgsea_1.30.0            clue_0.3-66             scales_1.3.0
#>  [91] png_0.1-8               ggfun_0.1.8             knitr_1.49
#>  [94] rstudioapi_0.17.1       tzdb_0.4.0              reshape2_1.4.4
#>  [97] rjson_0.2.23            nlme_3.1-166            curl_6.0.1
#> [100] org.Hs.eg.db_3.19.1     cachem_1.1.0            GlobalOptions_0.1.2
#> [103] parallel_4.4.2          restfulr_0.0.15         pillar_1.10.0
#> [106] vctrs_0.6.5             dbplyr_2.5.0            cluster_2.1.8
#> [109] evaluate_1.0.1          magick_2.8.5            tinytex_0.54
#> [112] GenomicFeatures_1.56.0  cli_3.6.3               locfit_1.5-9.10
#> [115] compiler_4.4.2          rlang_1.1.4             crayon_1.5.3
#> [118] labeling_0.4.3          interp_1.1-6            aroma.light_3.34.0
#> [121] plyr_1.8.9              fs_1.6.5                stringi_1.8.4
#> [124] mapplots_1.5.2          viridisLite_0.4.2       deldir_2.0-4
#> [127] munsell_0.5.1           lazyeval_0.2.2          GOSemSim_2.30.2
#> [130] Matrix_1.7-1            patchwork_1.3.0         hms_1.1.3
#> [133] bit64_4.5.2             KEGGREST_1.44.1         statmod_1.5.0
#> [136] igraph_2.1.2            memoise_2.0.1           ggtree_3.12.0
#> [139] fastmatch_1.1-6         bit_4.5.0.1             gson_0.1.0
#> [142] ape_5.8-1
```