OUTRIDER2

***Felix Brechtmann*[1], *Christian Mertes*[1], *Ines Scheller*[2], *Agne Matuseviciute*[1], *Vicente Yepez*[1,3], *Julien Gagneur*[1,2,3]**

[1] Technical University Munich, Department of Informatics, Munich, Germany
[2] Helmholtz Zentrum München, Institute of Computational Biology, Munich, Germany
[3] Quantitative Biosciences Munich, Gene Center, Ludwig-Maximilians Universität München, Munich, Germany

April 6, 2022

Abstract

In the field of diagnostics of rare diseases, RNA-seq is emerging as an important and complementary tool for whole exome and whole genome sequencing. *OUTRIDER* is a framework that detects aberrant gene expression within a group of samples. It uses the negative binomial distribution which is fitted for each gene over all samples. We additionally provide an autoencoder, which automatically controls for co-variation before fitting. After fitting, each sample can be tested for aberrantly expressed genes. Furthermore, *OUTRIDER* provides functionality to easily filter unexpressed genes, to analyse the data as well as to visualize the results.

If you use *OUTRIDER* in published research, please cite:

Brechtmann F*, Mertes C*, Matuseviciute A*, Yepez V, Avsec Z, Herzog M, Bader D M, Prokisch H, Gagneur J; **OUTRIDER: A statistical method for detecting aberrantly expressed genes in RNA sequencing data**; *AJHG*; 2018; DOI: https://doi.org/10.1016/j.ajhg.2018.10.025

Contents

1 Introduction

OUTRIDER (OUTlier in RNA-seq fInDER) is a tool for finding aberrantly expressed genes in RNA-seq samples. It does so by fitting a negative binomial model to RNA-seq read counts, correcting for variations in sequencing depth and apparent co-variations across samples. Read counts that significantly deviate from the distribution are detected as outliers. *OUTRIDER* makes use of an autoencoder to control automatically for confounders within the data. A scheme of this approach is given in Figure 1.

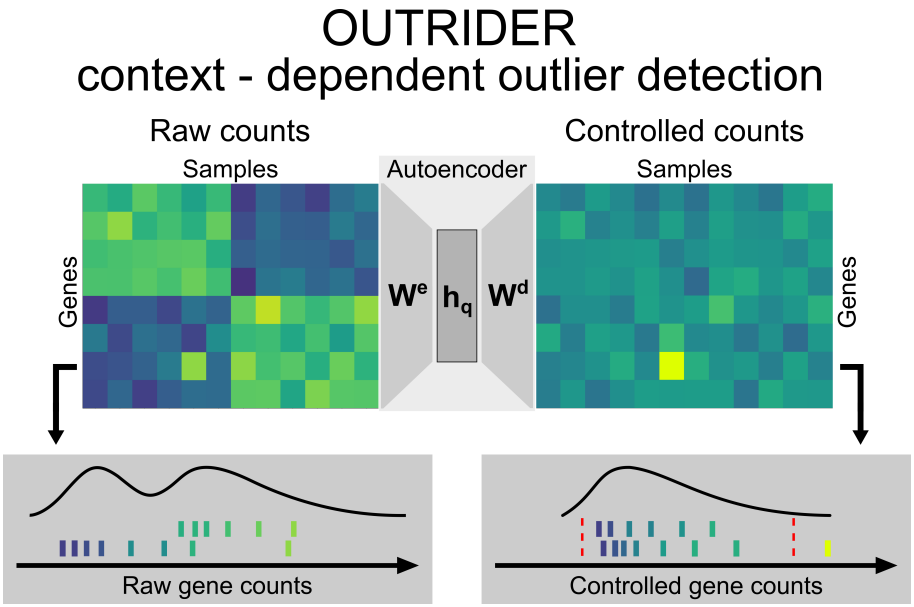


Figure 1: Context-dependent outlier detection. The algorithm identifies gene expression outliers whose read counts are significantly aberrant given the co-variations typically observed across genes in an RNA sequencing data set. This is illustrated by a read count (left panel, fifth column, second row from the bottom) that is exceptionally high in the context of correlated samples (left six samples) but not in absolute terms for this given gene. To capture commonly seen biological and technical contexts, an autoencoder models co-variations in an unsupervised fashion and predicts read count expectations. By comparing the earlier mentioned read count with these context-dependent expectations, it is revealed as exceptionally high (right panel). The lower panels illustrate the distribution of read counts before and after applying the correction for the relevant gene. The red dotted lines depict significance cutoffs.

Differential gene expression analysis from RNA-seq data is well-established. The packages *DESeq2*[1] or *edgeR*[2] provide effective workflows and preprocessing steps to perform differential gene expression analysis. However, these methods aim at detecting significant differences between groups of samples. In contrast, *OUTRIDER* aims at detecting outliers within a given population. A scheme of this difference is given in figure 2.

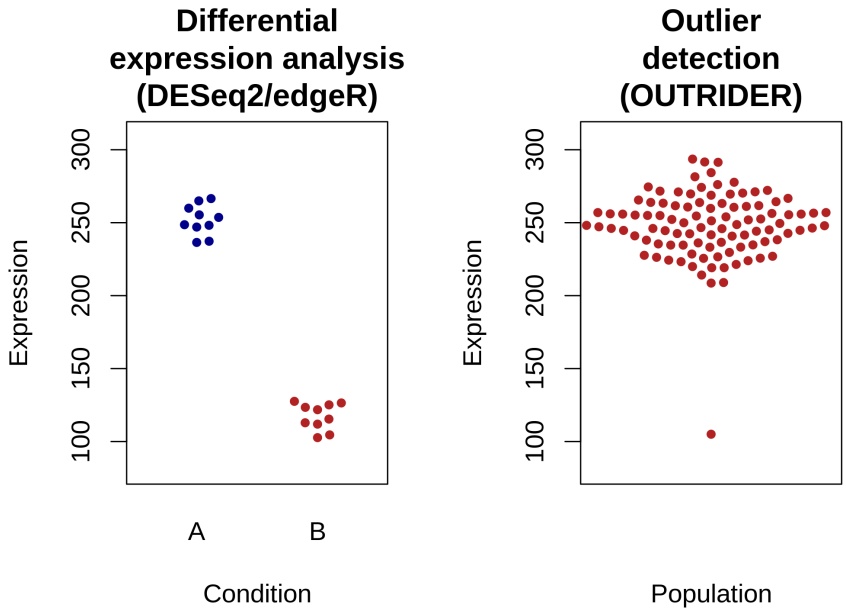


Figure 2: Scheme of workflow differences. Differences between differential gene expression analysis and outlier detection.

2 Prerequisites

To get started on the preprocessing step, we recommend to read the introductions of *DESeq2*[1], *edgeR*[2] or the RNA-seq workflow from Bioconductor: *rnaseqGene*. In brief, one usually starts with the raw FASTQ files from the RNA sequencing run. Those are then aligned to a given reference genome. At the time of writing (October 2018), we recommend the STAR aligner[3]. After obtaining the aligned BAM files, one can map the reads to exons or genes of a GTF annotation file using HT-seq or by using `summarizedOverlaps` from *GenomicAlignments*. The resulting count table can then be loaded into the *OUTRIDER* package as we will describe below.

3 A quick tour

Here we assume that we already have a count table and no additional preprocessing needs to be done. We can start and obtain results with 3 commands. First, create an *OutriderDataSet* from a count table or a Summarized Experiment object. Second, run the full pipeline using the command `OUTRIDER`. In the third and last step the results table is extracted from the *OutriderDataSet* with the `results` function. Furthermore, analysis plots that are described in section 5 can be directly created from the *OutriderDataSet* object.

```
library(OUTRIDER)
```

```
# get data
ctsFile <- system.file('extdata', 'KremerNBaderSmall.tsv',
        package='OUTRIDER')
ctsTable <- read.table(ctsFile, check.names=FALSE)
ods <- OutriderDataSet(countData=ctsTable)

# filter out non expressed genes
ods <- filterExpression(ods, minCounts=TRUE, filterGenes=TRUE)

# run full OUTRIDER pipeline (control, fit model, calculate P-values)
ods <- OUTRIDER(ods)
```

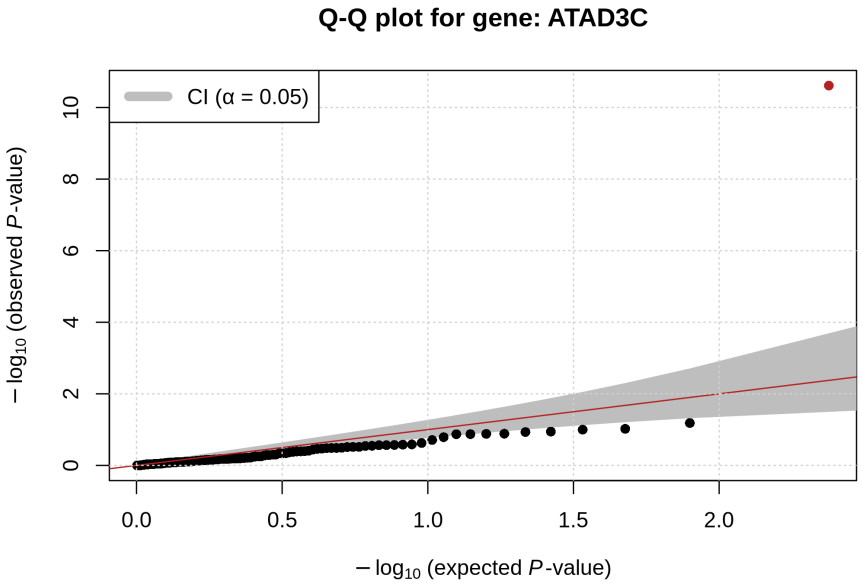
```
## [1] "Wed Apr  6 17:10:30 2022: Initial PCA loss: 4.74152408045355"
## [1] "Wed Apr  6 17:10:44 2022: Iteration: 1 loss: 4.19104023483289"
## [1] "Wed Apr  6 17:10:50 2022: Iteration: 2 loss: 4.17136841459552"
## [1] "Wed Apr  6 17:10:56 2022: Iteration: 3 loss: 4.16242004566665"
## [1] "Wed Apr  6 17:11:03 2022: Iteration: 4 loss: 4.15809323248252"
## [1] "Wed Apr  6 17:11:09 2022: Iteration: 5 loss: 4.15499133191828"
## [1] "Wed Apr  6 17:11:15 2022: Iteration: 6 loss: 4.15323016083236"
## [1] "Wed Apr  6 17:11:21 2022: Iteration: 7 loss: 4.15150191357432"
## [1] "Wed Apr  6 17:11:27 2022: Iteration: 8 loss: 4.15025067762997"
## [1] "Wed Apr  6 17:11:33 2022: Iteration: 9 loss: 4.14990682948401"
## [1] "Wed Apr  6 17:11:39 2022: Iteration: 10 loss: 4.14983030340635"
## [1] "Wed Apr  6 17:11:45 2022: Iteration: 11 loss: 4.14975819844699"
## [1] "Wed Apr  6 17:11:50 2022: Iteration: 12 loss: 4.14968525545797"
## [1] "Wed Apr  6 17:11:56 2022: Iteration: 13 loss: 4.14906750472369"
## [1] "Wed Apr  6 17:12:02 2022: Iteration: 14 loss: 4.14864152476243"
## [1] "Wed Apr  6 17:12:08 2022: Iteration: 15 loss: 4.14847213196995"
## Time difference of 1.524078 mins
## [1] "Wed Apr  6 17:12:08 2022: 15 Final nb-AE loss: 4.14847213196995"
```

```
# results (only significant)
res <- results(ods)
head(res)
```

```
##    geneID sampleID       pValue      padjust zScore   fc log2fc rawcounts
## 1: ATAD3C  MUC1360 2.444746e-11 1.359933e-07   5.33 3.75   1.91       948
## 2:  MSTO1  MUC1367 2.280086e-09 1.268338e-05  -6.30 0.57  -0.82       761
## 3: NBPF15  MUC1351 3.002774e-09 1.670346e-05   5.55 1.67   0.74      7591
## 4:  HDAC1  MUC1350 1.138155e-08 6.331186e-05  -5.98 0.58  -0.78      2215
## 5:  DCAF6  MUC1374 5.558464e-08 3.091994e-04  -5.72 0.65  -0.61      2348
## 6: NBPF16  MUC1351 4.900810e-07 1.363081e-03   4.70 1.57   0.65      4014
##    expected_counts normcounts meanCorrected  theta sizefactor
## 1:          251.90     253.62         78.70  15.85       1.09
## 2:         1340.39     724.65       1244.93 154.01       1.01
## 3:         4555.09    6867.63       4363.06 114.23       1.10
```

```
## 4:         3815.84    2118.06       3588.06 138.31       1.13
## 5:         3590.18    3089.58       4582.92 201.87       0.87
## 6:         2550.48    3780.85       2550.76 107.74       1.10
##    pvalDistribution aberrant AberrantBySample AberrantByFeature padj_rank
## 1:               nb     TRUE                1                1         1
## 2:               nb     TRUE                1                1         1
## 3:               nb     TRUE                2                1         1
## 4:               nb     TRUE                1                1         1
## 5:               nb     TRUE                1                1         1
## 6:               nb     TRUE                2                1         2
```

```
# example of a Q-Q plot for the most significant outlier
plotQQ(ods, res[1, geneID])
```



4 An *OUTRIDER* analysis in detail

Apart from running the full pipeline using the single wrapper function `OUTRIDER`, the analysis can also be run step by step. The wrapper function does not include any preprocessing functions. Discarding non expressed genes or samples failing quality measurements should be done manually before running the `OUTRIDER` function or starting the analysis pipeline.

In this section we will explain the analysis functions step by step.

For this tutorial we will use the rare disease data set from Kremer *et al.*[4]. For testing purposes, this package contains a small subset of it.

4.1 OutriderDataSet

To use *OUTRIDER* create an *OutriderDataSet*, which derives from a RangedSummarizedExperiment object. The *OutriderDataSet*can be created by supplying a count matrix and optional sample annotation matrices. Alternatively, an existing Summarized experiment object from other Biocnductor backages can be used.

```
# small testing data set
odsSmall <- makeExampleOutriderDataSet(dataset="Kremer")

# full data set from Kremer et al.
baseURL <- paste0("https://static-content.springer.com/esm/",
        "art%3A10.1038%2Fncomms15824/MediaObjects/")
count_URL <- paste0(baseURL, "41467_2017_BFncomms15824_MOESM390_ESM.txt")
anno_URL  <- paste0(baseURL, "41467_2017_BFncomms15824_MOESM397_ESM.txt")

ctsTable <- read.table(count_URL, sep="\t")
annoTable <- read.table(anno_URL, sep="\t", header=TRUE)
annoTable$sampleID <- annoTable$RNA_ID

# create OutriderDataSet object
ods <- OutriderDataSet(countData=ctsTable, colData=annoTable)
```

4.2 Preprocessing

It is recommended to preprocess the data before fitting. Our model requires that for every gene at least one sample has a non-zero count and that we observe at least one read for every 100 samples. Therefore, all genes that are not expressed must be discarded.

We provide the function `filterExpression` to remove genes that have low FPKM (Fragments Per Kilobase of transcript per Million mapped reads) expression values. The needed annotation to estimate FPKM values from the counts should be the same as for the counting. Here, we normalize by the total exon length of a gene. To do so the joint lenght of all exons needs to be provided. When providing a gtf, gff or TxDb object to the `filterExpression`, we extract this information automatically. But therfore the geneID's of the count table and the gtf need to match.

By default the cutoff is set to an FPKM value of one and only the filtered *OutriderDataSet* object is returned. If required, the FPKM values can be stored in the *OutriderDataSet* object and the full object can be returned to visualize the distribution of reads before and after filtering.

```
# get annotation
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(org.Hs.eg.db)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
```

```
map <- select(org.Hs.eg.db, keys=keys(txdb, keytype = "GENEID"),
        keytype="ENTREZID", columns=c("SYMBOL"))
```

However, the `TxDb.Hsapiens.UCSC.hg19.knownGene` contains only well annotated genes. This annotation will miss a lot of genes captured by RNA-seq. To include all predicted annotations as well as non-coding RNAs please download the txdb object from our homepage [1] or create it yourself from the UCSC website [2, 3].

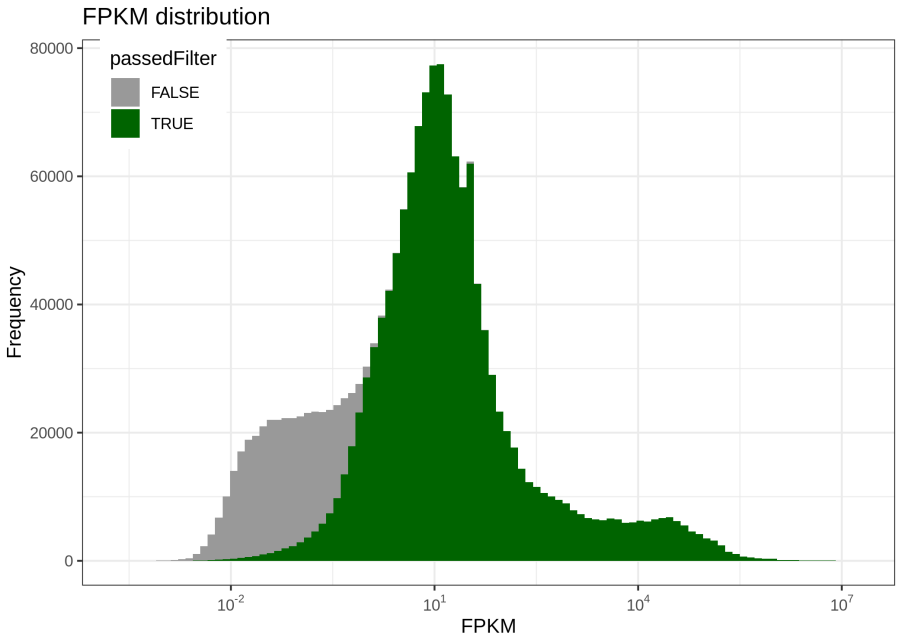
```
try({
    library(RMariaDB)
    library(AnnotationDbi)
    con <- dbConnect(MariaDB(), host='genome-mysql.cse.ucsc.edu',
            dbname="hg19", user='genome')
    map <- dbGetQuery(con, 'select kgId AS TXNAME, geneSymbol from kgXref')

    txdbUrl <- paste0("https://cmm.in.tum.de/public/",
            "paper/mitoMultiOmics/ucsc.knownGenes.db")
    download.file(txdbUrl, "ucsc.knownGenes.db")
    txdb <- loadDb("ucsc.knownGenes.db")

})
```

```
# calculate FPKM values and label not expressed genes
ods <- filterExpression(ods, txdb, mapping=map,
        filterGenes=FALSE, savefpkm=TRUE)

# display the FPKM distribution of counts.
plotFPKM(ods)
```

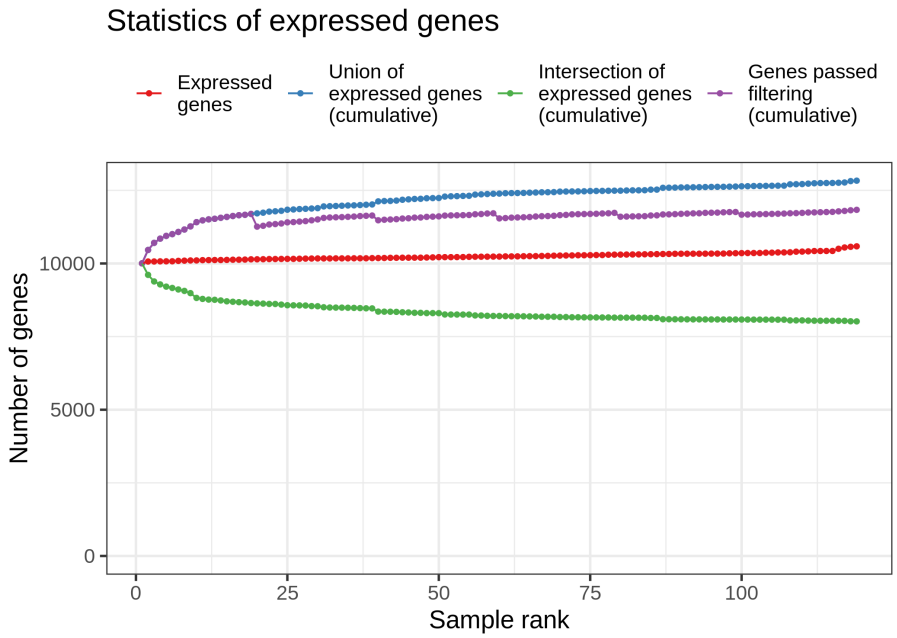
[1] https://cmm.in.tum.de/public/paper/mitoMultiOmics/ucsc.knownGenes.db

[2] https://genome.ucsc.edu/cgi-bin/hgTables

[3] http://genomewiki.ucsc.edu/index.php/Genes_in_gtf_or_gff_format




```
# display gene filter summary statistics
plotExpressedGenes(ods)
```

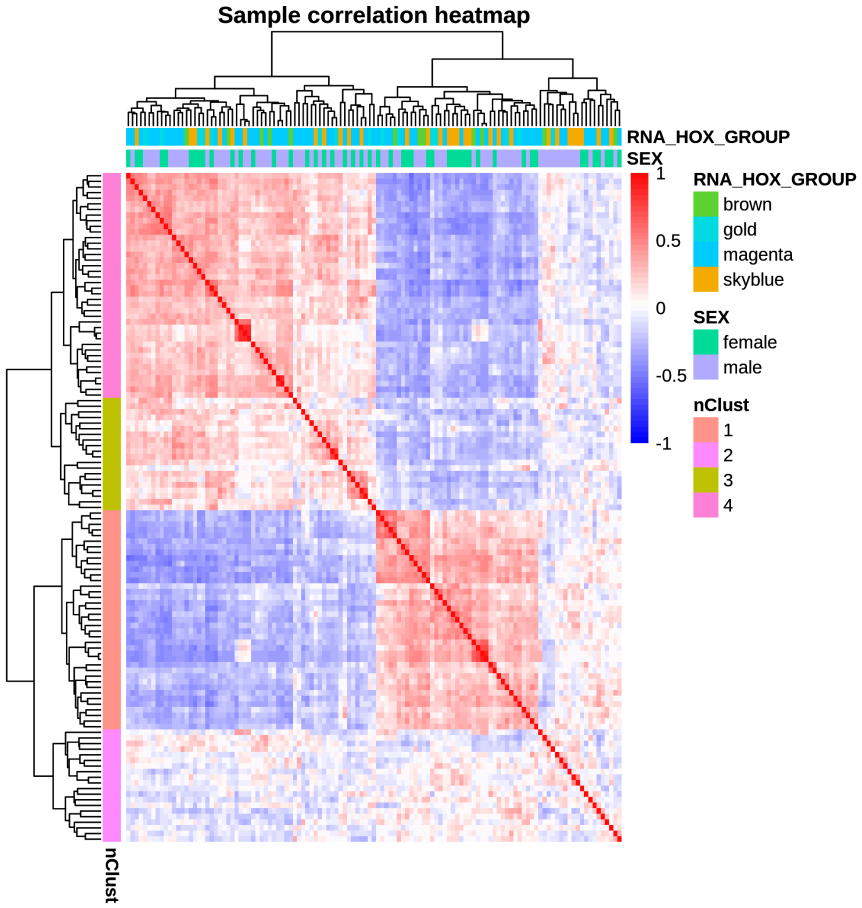



```
# do the actual subsetting based on the filtering labels
ods <- ods[mcols(ods)$passedFilter,]
```

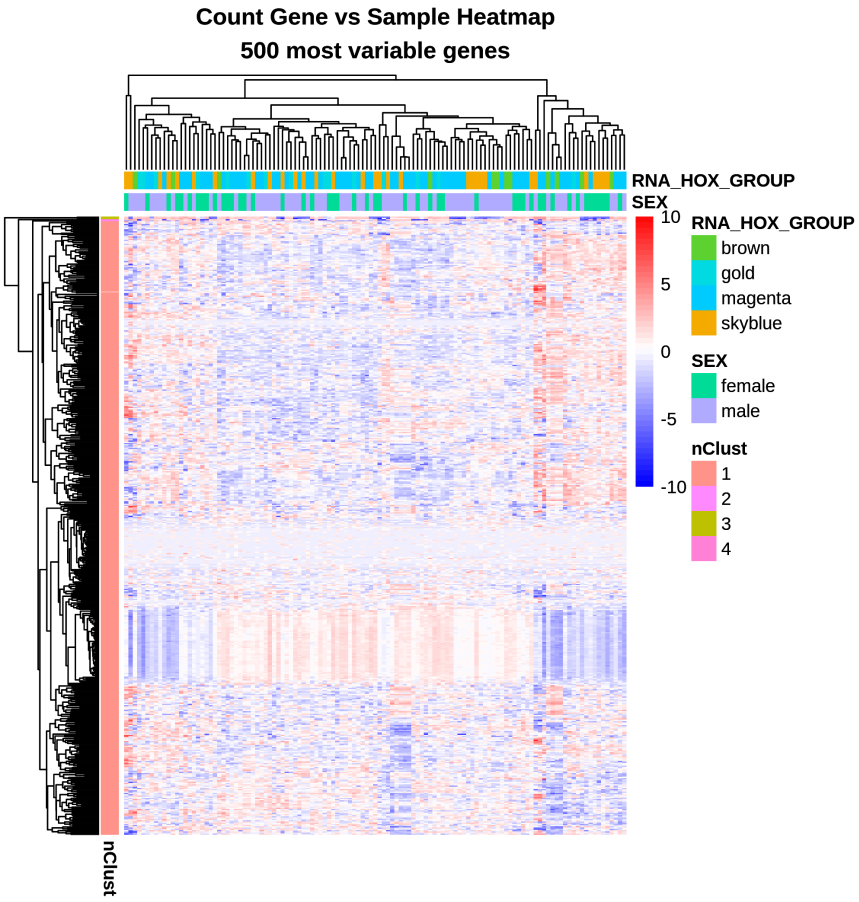
4.3 Controlling for Confounders

The next step in any analysis workflow is to visualize the correlations between samples. In most RNA-seq experiments correlations between the samples can be observed. These are often due to technical confounders (e.g. sequencing batch) or biological confounders (e.g. sex, age). These confounders can adversely affect the detection of aberrant features. Therefore, we provide options to control for them.

```
# Heatmap of the sample correlation
# it can also annotate the clusters resulting from the dendrogram
ods <- plotCountCorHeatmap(ods, colGroups=c("SEX", "RNA_HOX_GROUP"),
        normalized=FALSE, nRowCluster=4)
```




```
# Heatmap of the gene/sample expression
ods <- plotCountGeneSampleHeatmap(ods, colGroups=c("SEX", "RNA_HOX_GROUP"),
        normalized=FALSE, nRowCluster=4)
```



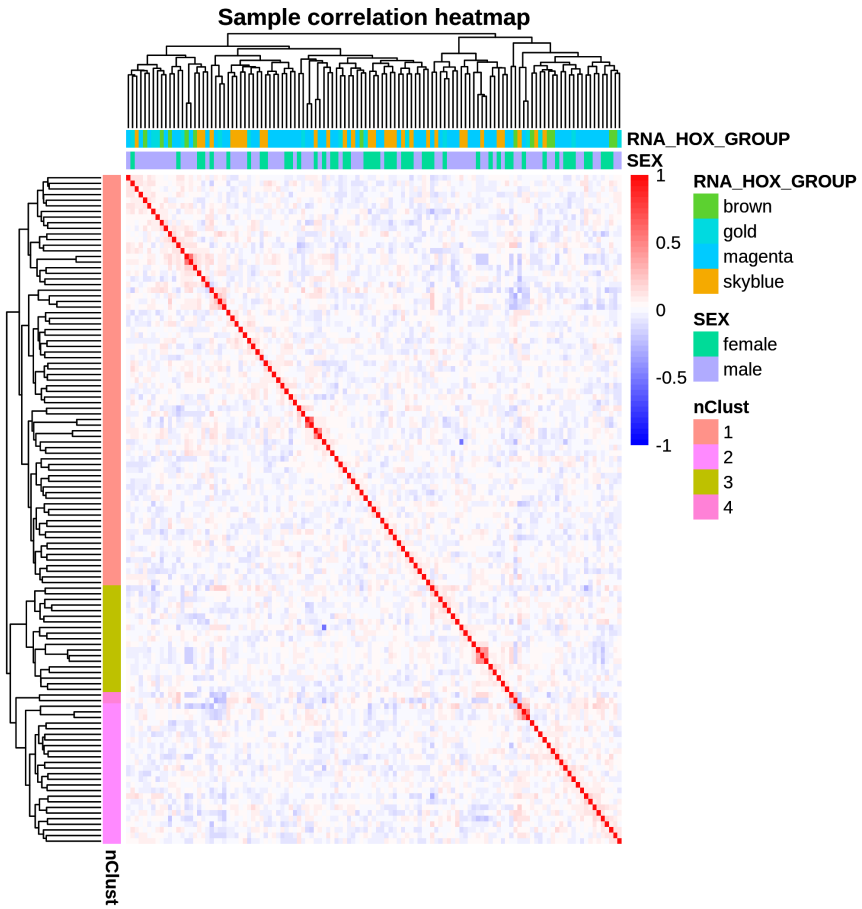
We have different ways to control for confounders present in the data. The first and standard way is to calculate the `sizeFactors` as done in *DESeq2*[1].

Additionally, the `controlForConfounders` function calls an autoencoder that automatically controls for confounders present in the data. Therefore an encoding dimension q needs to be set or the default value 20 is used. The optimal value of q can be determined using the `findEncodingDim` function. After controlling for confounders, the heatmap should be plotted again. If it worked, no batches should be present and the correlations between samples should be reduced and close to zero.

```
# automatically control for confounders
# we use only 3 iterations to make the vignette faster. The default is 15.
ods <- preprocess(ods) # estimates sizeFactors
ods <- controlForConfounders(ods, q=21, iterations=3)
```

```
## [1] "Wed Apr  6 17:12:55 2022: Initial PCA loss: 5.97224827481519"
## [1] "Wed Apr  6 17:16:26 2022: Iteration: 1 loss: 5.39030655165018"
## [1] "Wed Apr  6 17:18:11 2022: Iteration: 2 loss: 5.37689216747928"
## [1] "Wed Apr  6 17:19:53 2022: Iteration: 3 loss: 5.3706326055923"
## Time difference of 5.366931 mins
## [1] "Wed Apr  6 17:19:53 2022: 3 Final nb-AE loss: 5.3706326055923"
```

```
# Heatmap of the sample correlation after controlling
ods <- plotCountCorHeatmap(ods, normalized=TRUE,
        colGroups=c("SEX", "RNA_HOX_GROUP"))
```



Alternatively, other methods can be used to control for confounders. In addition to the *autoencoder*, we implemented a PCA based approach. The PCA implementation can be utilized by setting `implementation="pca"`. Also PEER can be used together with the OUTRIDER framework. A detailed description on how to do this can be found in section 6.1. Furthermore, any other method can be used by providing the `normalizationFactor` matrix. This matrix must be computed beforehand using the appropriate method. Its purpose is to normalize for technical effects or control for additional expression patterns.

4.4 Finding the right encoding dimension q

In the previous section, we fixed the encoding dimension $q = 21$. But having the right encoding dimension is crucial in finding outliers in the data. On the one hand, if q is too big the autoencoder will learn the identity matrix and will overfit the data. On the other hand, if q is too small the autoencoder cannot learn the necessary

covariates existing in the data. Therefore, it is recommended for any new dataset to estimate the optimal encoding dimension to gain the best performance. With the function `findEncodingDim` one can find the optimal encoding dimension. To this end, we artificially introduce corrupted counts randomly into the dataset and monitor the performance calling those corrupted counts. The optimal dimension q is then selected as the dimension maximizing the area under the precision-recall curve for identifying corrupted counts.

```
# find the optimal encoding dimension q
ods <- findEncodingDim(ods)

# visualize the hyper parameter optimization
plotEncDimSearch(ods)
```

Since this function runs a full OUTRIDER fit for a range of encoding dimensions, it is quite CPU intensive, but can increase the overall performance of the autoencoder and is recommended for any data set. If q is not provided by the user, it will be estimated based on the number of samples.

4.4.1 Excluding samples from the autoencoder fit

Since OUTRIDER expects that each sample within the population is independent of all others, replicates could mask effects specific to this sample. This is also true if trios are present in the data, where the parents can be seen as biological replicates. Here, we recommend to exclude the sample of interest or the replicates from the fitting. Later on, for all samples P-values are calculated.

In this rare disease data set we know that two samples (MUC1344 and MUC1365) have the same defect. To exclude one or both of them, we can use the `sampleExclusionMask` function.

```
# set exclusion mask
sampleExclusionMask(ods) <- FALSE
sampleExclusionMask(ods[,"MUC1365"]) <- TRUE

# check which samples are excluded from the autoencoder fit
sampleExclusionMask(ods)
```

```
##   35834   57415   61695   61982   65937   66623   69245   69248   69456
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
##   70038   70041   72748   74123   74172   76619   76620   76621   76622
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
##   76623   76624   76625   76626   76627   76628   76629   76630   76631
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
##   76632   76633   76635   76636   76637   76638 MUC0486 MUC0487 MUC0488
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## MUC0489 MUC0490 MUC0491 MUC1342 MUC1343 MUC1344 MUC1345 MUC1346 MUC1347
```

```
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## MUC1348 MUC1349 MUC1350 MUC1351 MUC1352 MUC1354 MUC1355 MUC1357 MUC1358
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## MUC1359 MUC1360 MUC1361 MUC1362 MUC1363 MUC1364 MUC1365 MUC1367 MUC1368
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE    TRUE   FALSE   FALSE
## MUC1369 MUC1370 MUC1371 MUC1372 MUC1373 MUC1374 MUC1375 MUC1376 MUC1377
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## MUC1378 MUC1379 MUC1380 MUC1381 MUC1382 MUC1383 MUC1384 MUC1390 MUC1391
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## MUC1392 MUC1393 MUC1394 MUC1395 MUC1396 MUC1397 MUC1398 MUC1400 MUC1401
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## MUC1402 MUC1403 MUC1404 MUC1405 MUC1407 MUC1408 MUC1409 MUC1410 MUC1411
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## MUC1412 MUC1413 MUC1414 MUC1415 MUC1416 MUC1417 MUC1418 MUC1419 MUC1420
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## MUC1421 MUC1422 MUC1423 MUC1424 MUC1425 MUC1426 MUC1427 MUC1428 MUC1429
##   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE   FALSE
## MUC1436 MUC1437
##   FALSE   FALSE
```

4.5 Fitting the negative binomial model

The fit of the negative binomial model is done during the autoencoder fitting. This step is only needed if alternative methods to control the data are used. To fit the dispersion and the mean, the `fit` function is applied to the *OutriderDataSet*.

```
# fit the model when alternative methods where used in the control step
ods <- fit(ods)
hist(theta(ods))
```

4.6 P-value calculation

After determining the fit parameters, two-sided P-values are computed using the following equation:

$$p_{ij} = 2 \cdot min \left\{ \frac{1}{2}, \sum_{0}^{k_{ij}} NB(\mu_{ij}, \theta_i), 1 - \sum_{0}^{k_{ij}-1} NB(\mu_{ij}, \theta_i) \right\}, \quad (1)$$

where the $\frac{1}{2}$ term handles the case of both terms exceeding 0.5, which can happen due to the discrete nature of counts. Here μ_{ij} are computed as the product of the fitted correction values from the autoencoder and the fitted mean adjustements. If required a one-sided test can be performed using the argument `alternative` and specifying 'less' or 'greater' depending on the research question. Multiple testing

correction is done across all genes in a per-sample fashion using Benjamini-Yekutieli's false discovery rate method[5]. Alternatively, all adjustment methods supported by `p.adjust` can be used via the `method` argument.

```
# compute P-values (nominal and adjusted)
ods <- computePvalues(ods, alternative="two.sided", method="BY")
```

4.7 Z-score calculation

The Z-scores on the log transformed counts can be used for visualization, filtering, and ranking of samples. By running the `computeZscores` function, the Z-scores are computed and stored in the *OutriderDataSet* object. The Z-scores are calculated using:

$$z_{ij} = \frac{l_{ij} - \mu_j^l}{\sigma_j^l} \qquad (2)$$

$$l_{ij} = \log_2 (\frac{k_{ij} + 1}{c_{ij} + 1}),$$

where μ_j^l is the mean and σ_j^l the standard deviation of gene j and l_{ij} is the log transformed count after correction for confounders.

```
# compute the Z-scores
ods <- computeZscores(ods)
```

5 Results

The *OUTRIDER* package offers multiple ways to display the results. It creates a results table containing all the values computed during the analysis. Furthermore, it offers various plot functions that guide the user through the analysis.

5.1 Results table

The `results` function gathers all the previously computed values and combines them into one table.

```
# get results (default only significant, padj < 0.05)
res <- results(ods)
head(res)
```

```
##     geneID sampleID       pValue      padjust zScore rawcounts
## 1:  NUDT12    65937 2.518040e-22 2.951013e-17 -10.33         0
## 2:   STAG2  MUC0490 3.348034e-21 3.923723e-16  -9.74       622
## 3:  TALDO1  MUC1427 6.621510e-18 7.760068e-13  -9.44       482
```

```
## 4:  NLGN4Y  MUC1401 3.162343e-16 3.706103e-11   4.74        18
## 5: NDUFA10  MUC1358 2.844299e-15 1.666686e-10  -8.29      1137
## 6: CSNK2A1  MUC1358 2.720292e-15 1.666686e-10  -8.12      2060
##    expected_counts normcounts meanCorrected  theta sizefactor
## 1:          249.46       0.00        461.94  19.06       1.01
## 2:         2288.63    1105.44       4025.31  83.28       0.70
## 3:         4720.55     465.61       4547.84  27.77       1.11
## 4:            0.87    1412.43         95.35  33.48       1.10
## 5:         2485.05    1067.83       2350.19 141.36       1.16
## 6:         3279.81    2015.66       3213.41 383.53       1.16
##    pvalDistribution aberrant AberrantBySample AberrantByFeature padj_rank
## 1:               nb     TRUE                1                1       1.0
## 2:               nb     TRUE                6                1       1.0
## 3:               nb     TRUE                1                1       1.0
## 4:               nb     TRUE                2                2       1.0
## 5:               nb     TRUE                5                1       1.5
## 6:               nb     TRUE                5                1       1.5
```

```
dim(res)
```

```
## [1] 253  16
```

```
# setting a different significance level and filtering by Z-scores
res <- results(ods, padjCutoff=0.1, zScoreCutoff=2)
head(res)
```

```
##     geneID sampleID       pValue      padjust zScore rawcounts
## 1:  NUDT12    65937 2.518040e-22 2.951013e-17 -10.33         0
## 2:   STAG2  MUC0490 3.348034e-21 3.923723e-16  -9.74       622
## 3:  TALDO1  MUC1427 6.621510e-18 7.760068e-13  -9.44       482
## 4:  NLGN4Y  MUC1401 3.162343e-16 3.706103e-11   4.74        18
## 5: NDUFA10  MUC1358 2.844299e-15 1.666686e-10  -8.29      1137
## 6: CSNK2A1  MUC1358 2.720292e-15 1.666686e-10  -8.12      2060
##    expected_counts normcounts meanCorrected  theta sizefactor
## 1:          249.46       0.00        459.87  19.06       1.01
## 2:         2288.63    1105.44       4016.50  83.28       0.70
## 3:         4720.55     465.61       4569.28  27.77       1.11
## 4:            0.87    1412.43         91.20  33.48       1.10
## 5:         2485.05    1067.83       2351.03 141.36       1.16
## 6:         3279.81    2015.66       3214.79 383.53       1.16
##    pvalDistribution aberrant AberrantBySample AberrantByFeature padj_rank
## 1:               nb     TRUE                2                1       1.0
## 2:               nb     TRUE                6                1       1.0
## 3:               nb     TRUE                1                1       1.0
## 4:               nb     TRUE                2                2       1.0
## 5:               nb     TRUE                5                1       1.5
## 6:               nb     TRUE                5                1       1.5
```

```
dim(res)
```

```
## [1] 318  16
```

5.2 Number of aberrant genes per sample

One quantity of interest is the number of aberrantly expressed genes per sample. This can be displayed using the plotting function `plotAberrantPerSample`. Alternatively, the function `aberrant` can be used to identify aberrant events, which can be summed by sample or gene using the paramter `by`. These numbers depend on the cutoffs, which can be specified in both functions (`padjCutoff` and `zScoreCutoff`).

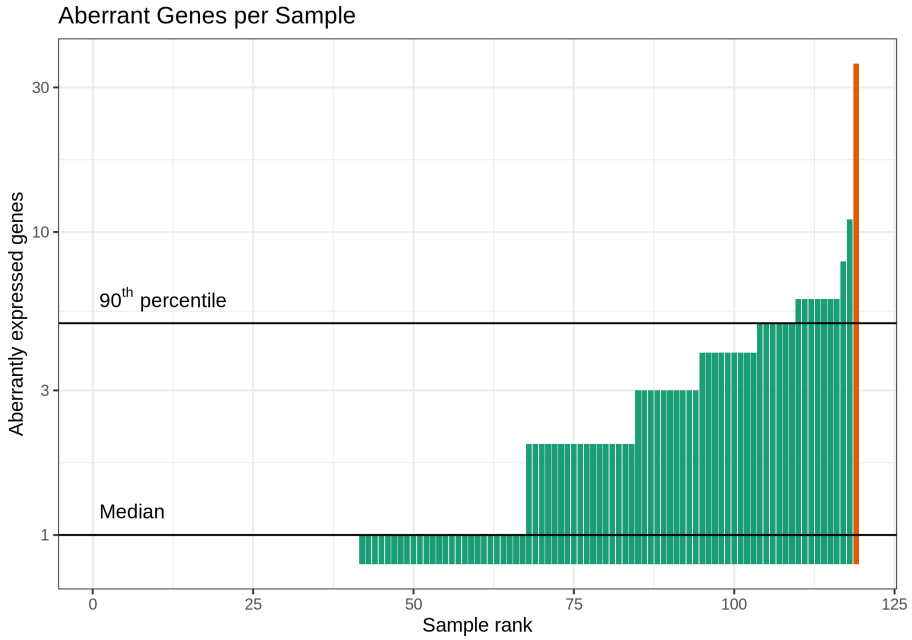
```
# number of aberrant genes per sample
tail(sort(aberrant(ods, by="sample")))
```

```
## MUC1342 MUC1367 MUC1381 MUC1363 MUC1364   76633
##       6       6       6       8      11      36
```

```
tail(sort(aberrant(ods, by="gene", zScoreCutoff=1)))
```

```
## HOXA10-HOXA9         ZFAT       DNAJC3  SLMO2-ATP5E         PRKY
##            2            2            2            2            2
##       NLGN4Y
##            2
```

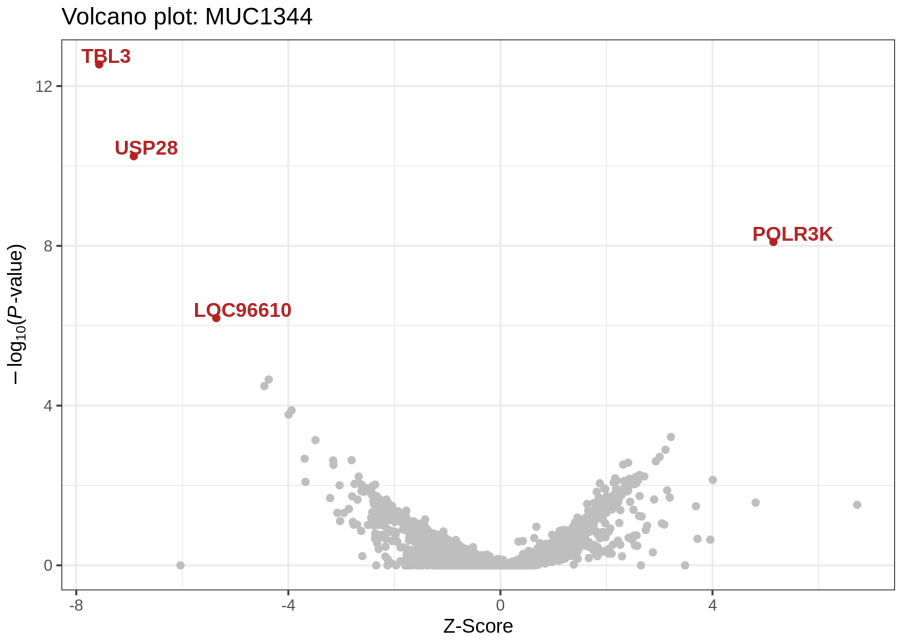
```
# plot the aberrant events per sample
plotAberrantPerSample(ods, padjCutoff=0.05)
```



5.3 Volcano plots

To view the distribution of P-values on a sample level, volcano plots can be displayed. Most of the plots make use of the *plotly* framework to create interactive plots. To only use basic R functionality from *graphics* the `basePlot` argument can be set to `TRUE`.

```
# MUC1344 is a diagnosed sample from Kremer et al.
plotVolcano(ods, "MUC1344", basePlot=TRUE)
```

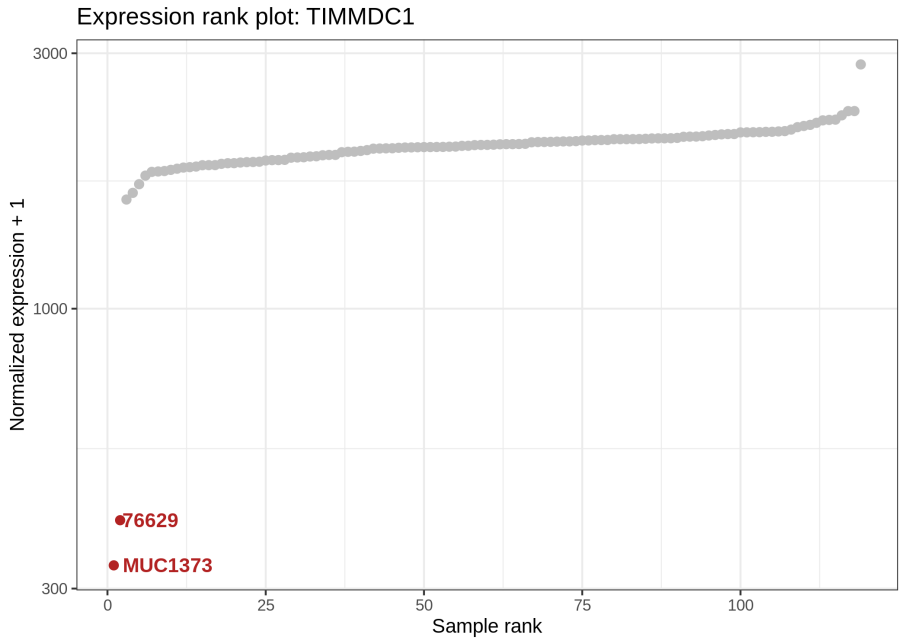


5.4 Gene level plots

Additionally, we include two plots at the gene level. `plotExpressionRank` plots the counts in ascending order. By default, the controlled counts are plotted. To plot raw counts, the argument `normalized` can be set to `FALSE`.

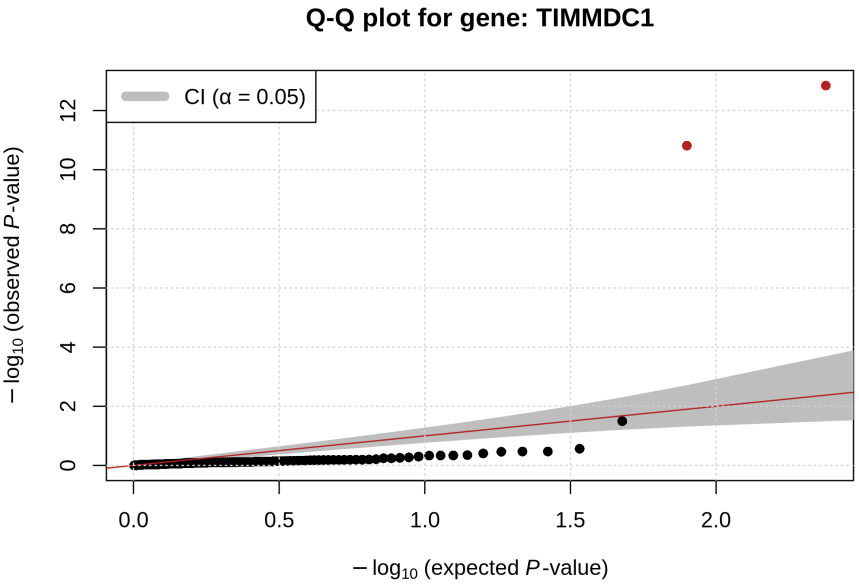
When using the *plotly* framework for plotting, all computed values are displayed for each data point. The user can access this information by hovering over each data point with the mouse.

```
# expression rank of a gene with outlier events
plotExpressionRank(ods, "TIMMDC1", basePlot=TRUE)
```



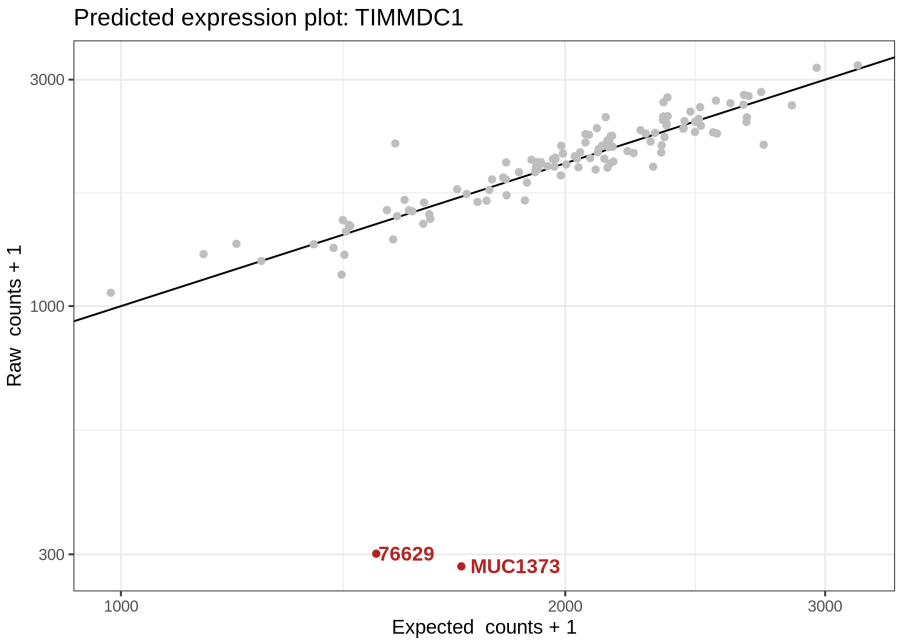
The quantile-quantile plot can be used to see whether the fit converged well. In presence of an outlier, it can happen that most of the points end up below the confidence band. This is fine and indicates that we have conservative P-values for the other points. Here is an example with two outliers:

```
## QQ-plot for a given gene
plotQQ(ods, "TIMMDC1")
```



Since we do test how fare the observed count is away from the exprected expression level, it is also helpful to visualize the predictions against the observed counts.

```
## Observed versus expected gene expression
plotExpectedVsObservedCounts(ods, "TIMMDC1", basePlot=TRUE)
```



6 Additional features

6.1 Using PEER to control for confounders

PEER[6] is a well known tool to control for unknown effects in RNA-seq data. PEER is only available through the *peer* GitHub repository. The R source code can be downloaded form here: https://github.com/downloads/PMBio/peer/R_peer_source_1.3.tgz. The installation of the package has to be done manually by the user. After the installation one can use the following function to control for confounders with PEER.

```
#'
#' PEER implementation
#'
peer <- function(ods, maxFactors=NA, maxItr=1000){

    # check for PEER
    if(!require(peer)){
        stop("Please install the 'peer' package from GitHub to use this ",
                "functionality.")
```

```
    }

    # default and recommendation by PEER: min(0.25*n, 100)
    if(is.na(maxFactors)){
        maxFactors <- min(as.integer(0.25* ncol(ods)), 100)
    }

    # log counts
    logCts <- log2(t(t(counts(ods)+1)/sizeFactors(ods)))

    # prepare PEER model
    model <- PEER()
    PEER_setNmax_iterations(model, maxItr)
    PEER_setNk(model, maxFactors)
    PEER_setPhenoMean(model, logCts)
    PEER_setAdd_mean(model, TRUE)

    # run fullpeer pipeline
    PEER_update(model)

    # extract PEER data
    peerResiduals <- PEER_getResiduals(model)
    peerMean <- t(t(2^(logCts - peerResiduals)) * sizeFactors(ods))

    # save model in object
    normalizationFactors(ods) <- pmax(peerMean, 1E-8)
    metadata(ods)[["PEER_model"]] <- list(
            alpha     = PEER_getAlpha(model),
            residuals = PEER_getResiduals(model),
            W         = PEER_getW(model))

    return(ods)
}
```

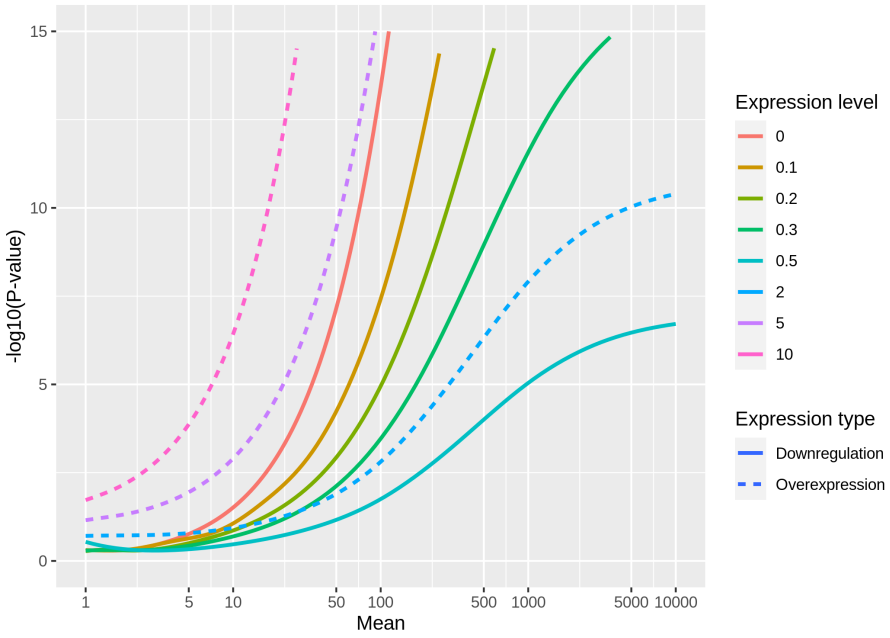
With the function above we can run the full OUTRIDER pipeline as follows:

```
# Control for confounders with PEER
ods <- estimateSizeFactors(ods)
ods <- peer(ods)
ods <- fit(ods)
ods <- computeZscores(ods, peerResiduals=TRUE)
ods <- computePvalues(ods)

# Heatmap of the sample correlation after controlling
ods <- plotCountCorHeatmap(ods, normalized=TRUE)
```

6.2 Power anaylsis

We provide the `plotPowerAnalysis` function to show, what kind of changes can be significant depending on the mean count.

```
## P-values versus Mean Count
plotPowerAnalysis(ods)
```



Here, we see that it is only for sufficiently high expressed genes possible, to obtain significant P-values, especially for the downregulation cases.

7 OUTRIDER2: a generalized framework for context-dependent outlier detection in omics data

Since version 2.0, *OUTRIDER* additionally supports model fitting and p value based outlier detection for the Gaussian distribution, as well as the inclusion of known confounders in the fitting and allows for the usage of different data preprocessing and transformation options. Use cases could for example, among others, be outlier detection in proteomics mass spectrometry intensities ('PROTRIDER'), or modelling vst transformed gene counts.

7.1 Outrider2DataSet

The *Outrider2DataSet* is a generalization of the *OutriderDataSet* class. Input values no longer have to be counts/integers, but can be continuous values and might also contain some missing values (NAs). When creating the *Outrider2DataSet*, the correct profile should be specified to indicate which default settings should be used

for preprocessing and model fitting. Available profiles are 'outrider' for the standart negative binomial OUTRIDER fit of gene counts, 'protrider' for fitting mass spectrometry intensities e.g. from proteomics measurements, and 'other' by default for a gaussian fit directly to the input data. To get more control over the underlying settings for each profile, see the next section on how to finetune the default parameter settings.

```
# simulate intensity data
n_samples <- 20
n_features <- 500
sim_intensities <- matrix(rlnorm(n_samples*n_features, meanlog=13, sdlog=2),
                          nrow=n_features, ncol=n_samples)
rownames(sim_intensities) <- paste0("feature_", seq_len(n_features))
colnames(sim_intensities) <- paste0("sample_", seq_len(n_samples))

# create Outrider2Dataset, here with 'protrider' profile:
ods <- Outrider2DataSet(inputData=sim_intensities, profile="protrider")
ods
```

```
## class: Outrider2DataSet
## class: RangedSummarizedExperiment
## dim: 500 20
## metadata(1): version
## assays(1): observed
## rownames(500): feature_1 feature_2 ... feature_499 feature_500
## rowData names(0):
## colnames(20): sample_1 sample_2 ... sample_19 sample_20
## colData names(1): sampleID
## ------------------ Model parameters ------------------
## Profile:                  protrider
## Default distribution:     gaussian
```

As *OutriderDataSet* is now a subclass of *Outrider2DataSet*, the profile is automatically set to 'outrider' when creating an *OutriderDataSet*.

7.2 Using the python backend

By default, fitting the R autoencoder implementation is used for the standard NB OUTRIDER fit without the inclusion of known confounders, unless the sample size is large ($>$ 1000 samples). In all other cases, the python backend is used for model fitting. The python backend to OUTRIDER is implemented in the 'py_outrider' package which can be installed from pip by running *pip install py_outrider*. *OUTRIDER*2 supports both manually specyfing a python binary or conda environment in which py_outrider has been installed, or letting the package automatically create an appropriate conda environment with the *basilisk* package

```
# default: using manually specified conda environment:
reticulate::use_condaenv("py_outrider")
ods <- OUTRIDER(ods)

# alternatively, let basilisk handle py_outrider installation:
ods <- OUTRIDER(ods, useBasilisk=TRUE)
```

7.3 Modifying the default data preprocessing options

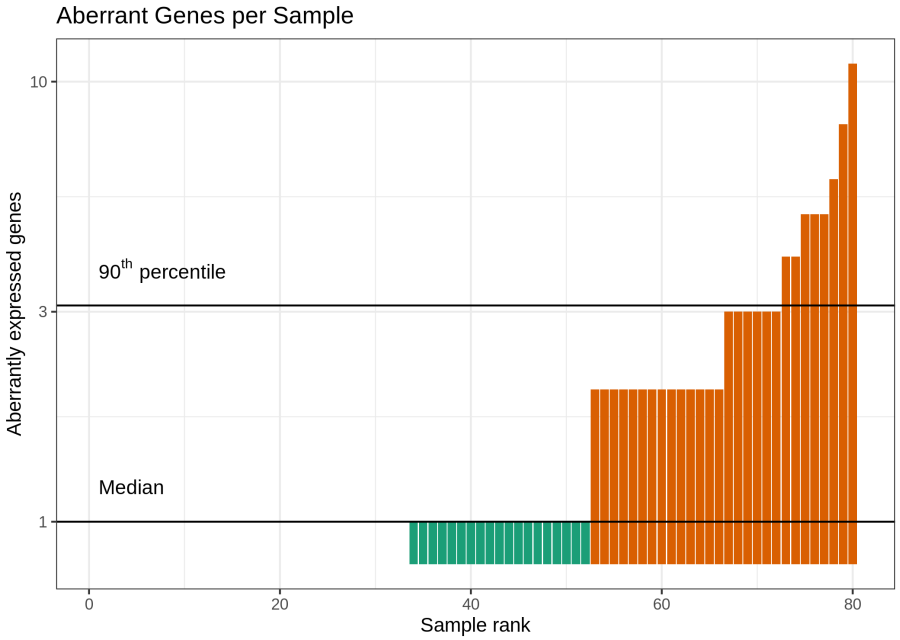
*OUTRIDER*2 supports the application of user specified data preprocessing functions prior to fitting. One example is the usage of a variance stabilizing transformation (vst) and then fitting the data with the Gaussian distribution. While this is typically faster than the NB autoencoder, it tends to produce a higher number of outliers per sample.

```
# create some example count data
ods <- makeExampleOutriderDataSet()

# set profile to 'other' to use general settings (uses gaussian distribution)
profile(ods) <- "other"

# modify preprocessing options, set 'prepro_func' to e.g. DESeq2 vst
prepro_opts <- getDefaultPreproParams(ods)
prepro_opts[["prepro_func"]] <- DESeq2::varianceStabilizingTransformation

# fit model with modified preprocessing options (requires python backend)
ods <- OUTRIDER(ods, prepro_options=prepro_opts, usePython=TRUE,
                useBasilisk=TRUE)

# check results
head(results(ods))
```

```
##         geneID  sampleID       pValue      padjust zScore delta input_value
## 1: feature_192 sample_47 2.193340e-11 2.578504e-08  -6.69 -4.40           1
## 2: feature_145 sample_37 1.026486e-10 1.206743e-07  -6.46 -8.20           0
## 3:  feature_43  sample_3 8.649540e-10 1.016845e-06  -6.13 -4.77           2
## 4:  feature_35 sample_15 1.224151e-09 1.439119e-06  -6.08 -7.26           0
## 5: feature_131 sample_44 1.520040e-09 1.786968e-06  -6.04 -7.58           0
## 6: feature_126 sample_32 1.721782e-09 2.024137e-06  -6.02 -8.04           0
##    preprocessed_raw preprocessed_expected normalized meanCorrected   sd
## 1:            -7.04                -2.64       1.73          6.03 3.31
## 2:           -11.66                -3.46      -4.61          3.53 3.77
## 3:            -5.61                -0.84       2.36          7.09 4.68
## 4:           -11.66                -4.40      -3.97          3.20 5.10
## 5:           -11.66                -4.08      -2.29          5.15 5.15
```

```
## 6:           -11.66                -3.62      -1.00          6.99 4.96
##    pvalDistribution aberrant AberrantBySample AberrantByFeature padj_rank
## 1:         gaussian     TRUE               5                 1         1
## 2:         gaussian     TRUE               2                 1         1
## 3:         gaussian     TRUE               2                 1         1
## 4:         gaussian     TRUE               1                 1         1
## 5:         gaussian     TRUE               2                 1         1
## 6:         gaussian     TRUE               1                 1         1
```

```
# plotting, e.g.
plotAberrantPerSample(ods)
```



In a analogous fashion, different data transformation functions to generate the autoencoder input, can be chosen with `prepro_opt[["data_trans"]] <- "log"`. Here, however, currently only 'log', 'log1p' or `NULL` (no transformation) are supported. Also, sizefactor normalization can be turned on or off by setting `prepro_opt[["sf_norm"]] <- TRUE/FALSE`. By default, the distribution specified in `prepro_opt[["distribution"]]` is used both for the model loss and for the p value calculation. At the moment, only 'nb' (negative binomial) and 'gaussian' are supported. A different distribution might be used for the model loss than for the p value calculation, in this case, the argument `loss_distribution` should be supplied to the `OUTRIDER` function to specify which distribution to use for the model loss.

7.4 Including known confounders in the model fitting

To explicitly consider confouders known beforehand, the user only has to specify the respective columns in the 'colData' of the *Outrider2DataSet*. The column will then be included in the model as a one-hot encoding, so it is only meaningful to specify columns that contain several distinct categories, not continuous values.

```
ods <- makeExampleProtriderDataSet()

# filter out features that are not variable across samples or have many NAs
ods <- filterExpression(ods)

# covariates that should be included in the fit have to be a column in colData
colData(ods)
```

```
## DataFrame with 80 rows and 8 columns
##              sampleID     batch trueSizeFactor expressedFeatures
##           <character> <numeric>      <numeric>         <numeric>
## sample_1     sample_1         2        1.01230               192
## sample_2     sample_2         1        1.05583               190
## sample_3     sample_3         4        1.18906               185
## sample_4     sample_4         2        1.08088               192
## sample_5     sample_5         2        1.03086               186
## ...               ...       ...            ...               ...
## sample_76   sample_76         3       1.156769               189
## sample_77   sample_77         4       1.078433               189
## sample_78   sample_78         4       0.995997               192
## sample_79   sample_79         2       1.028737               193
## sample_80   sample_80         4       0.910182               187
##           unionExpressedFeatures intersectionExpressedFeatures
##                        <numeric>                     <numeric>
## sample_1                     200                             7
## sample_2                     200                            17
## sample_3                     200                           151
## sample_4                     200                             7
## sample_5                     200                           142
## ...                          ...                           ...
## sample_76                    200                            21
## sample_77                    200                            18
## sample_78                    200                             6
## sample_79                    200                             5
## sample_80                    200                            54
##           passedFilterFeatures expressedFeaturesRank
##                      <numeric>             <integer>
## sample_1                   200                    52
## sample_2                   200                    34
```

```
## sample_3                  151                    3
## sample_4                  200                   53
## sample_5                  190                    4
## ...                       ...                  ...
## sample_76                 200                   32
## sample_77                 200                   33
## sample_78                 200                   63
## sample_79                 200                   73
## sample_80                 200                   17
```

```
# fit, including known confounders (here: 'batch')
ods <- OUTRIDER(ods, covariates = c("batch"), useBasilisk=TRUE)
```

7.5 Results table

The *OUTRIDER*2 results table is very similar to previous results table (see Section 5), but additionally specifies which distribution was used for obtaining the p values, and if preprocessing was applied, additionally lists the values after applying the preprocessing function. Furthermore, the default cutoffs on adjusted p values and effect size (fold change, zscores, or delta values, if appropriate) might be adjusted according to the user's preferences.

```
# default results extraction (p adjust < 0.05)
res <- results(ods)
head(res)
```

```
##      featureID  sampleID       pValue      padjust zScore   fc log2fc
## 1: feature_168 sample_74 2.442491e-15 2.637525e-12   7.91 1.21   0.27
## 2:   feature_2 sample_63 2.445487e-15 2.774268e-12  -7.92 0.81  -0.30
## 3: feature_107 sample_73 4.662937e-15 5.162361e-12   7.83 1.19   0.25
## 4:  feature_48 sample_56 5.919519e-15 6.472800e-12  -7.81 0.84  -0.25
## 5:  feature_33 sample_42 5.983925e-15 6.706569e-12  -7.80 0.85  -0.24
## 6: feature_160 sample_53 8.437695e-15 9.111450e-12   7.76 1.15   0.20
##    input_value preprocessed_raw preprocessed_expected normalized
## 1: 15758837.90            23.91                 19.83      22.92
## 2:   274360.72            18.07                 22.26      15.57
## 3: 17470836.26            24.06                 20.24      22.11
## 4:    23612.82            14.53                 17.29      15.92
## 5:    77710.11            16.25                 19.18      15.98
## 6: 27796680.24            24.73                 21.54      21.80
##    meanCorrected   sd sizefactor pvalDistribution aberrant AberrantBySample
## 1:         19.24 3.22       1.00         gaussian     TRUE                1
## 2:         18.95 3.13       1.01         gaussian     TRUE                1
## 3:         18.90 3.17       1.01         gaussian     TRUE                1
## 4:         18.72 1.90       1.00         gaussian     TRUE                1
```

```
## 5:         18.65 1.89       1.00         gaussian     TRUE                2
## 6:         19.26 2.25       1.02         gaussian     TRUE                1
##    AberrantByFeature padj_rank
## 1:                 1         1
## 2:                 1         1
## 3:                 1         1
## 4:                 1         1
## 5:                 1         1
## 6:                 1         1
```

```
# adjusting the cutoffs used for extracting results
res <- results(ods, padjCutoff=0.1, l2fcCutoff=0.5, zScoreCutoff=5)
head(res)
```

```
## Empty data.table (0 rows and 20 cols): featureID,sampleID,pValue,padjust,zScore,fc...
```

7.6 Plotting

The same plotting functions as described in Section 5 may be used to visualize *Outrider2DataSet* results. The name of some feature-level functions has been adapted to support the more general *Outrider2DataSet* objects:

- `plotExpectedVsObservedCounts` -> `plotExpectedVsObserved`,
- `plotCountCorHeatmap` -> `plotSampleCorHeatmap`,
- `plotCountGeneSampleHeatmap` -> `plotFeatureSampleHeatmap`.

References

[1] Michael I Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, 15(12):550, dec 2014. URL: http://genomebiology.biomedcentral.com/articles/10.1186/s13059-014-0550-8, doi:10.1186/s13059-014-0550-8.

[2] Mark D Robinson, Davis J. McCarthy, and Gordon K Smyth. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics (Oxford, England)*, 26(1):139–40, jan 2010. URL: https://doi.org/10.1093/bioinformatics/btp616, doi:10.1093/bioinformatics/btp616.

[3] Alexander Dobin, Carrie A. Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R. Gingeras. STAR: Ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013. URL: https://doi.org/10.1093/bioinformatics/bts635, doi:10.1093/bioinformatics/bts635.

[4] Laura S Kremer, Daniel M Bader, Christian Mertes, Robert Kopajtich, Garwin Pichler, Arcangela Iuso, Tobias B Haack, Elisabeth Graf, Thomas Schwarzmayr, Caterina Terrile, Eliška Koňaříková, Birgit Repp, Gabi Kastenmüller, Jerzy Adamski, Peter Lichtner, Christoph Leonhardt, Benoit Funalot, Alice Donati, Valeria Tiranti, Anne Lombes, Claude Jardel, Dieter Gläser, Robert W Taylor, Daniele Ghezzi, Johannes A Mayr, Agnes Rötig, Peter Freisinger, Felix Distelmaier, Tim M Strom, Thomas Meitinger, Julien Gagneur, and Holger Prokisch. Genetic diagnosis of Mendelian disorders via RNA sequencing. *Nature Communications*, 8:15824, jun 2017. URL: https://www.nature.com/articles/ncomms15824.pdf, doi:10.1038/ncomms15824.

[5] Yoav Benjamini and Daniel Yekutieli. The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics*, 29(4):1165–1188, 2001. URL: https://projecteuclid.org/euclid.aos/1013699998, arXiv:0801.1095, doi:10.1214/aos/1013699998.

[6] Oliver Stegle, Leopold Parts, Matias Piipari, John Winn, and Richard Durbin. Using probabilistic estimation of expression residuals (PEER) to obtain increased power and interpretability of gene expression analyses. *Nature Protocols*, 7(3):500–507, 2012. doi:10.1038/nprot.2011.457.

Session info

Here is the output of `sessionInfo()` on the system on which this document was compiled:

```
## R version 4.1.2 (2021-11-01)
## Platform: x86_64-conda-linux-gnu (64-bit)
## Running under: Ubuntu 21.10
##
## Matrix products: default
## BLAS/LAPACK: /home/ines/miniconda3/envs/r_4.1/lib/libopenblasp-r0.3.18.so
##
## locale:
##  [1] LC_CTYPE=de_DE.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=de_DE.UTF-8        LC_COLLATE=de_DE.UTF-8
##  [5] LC_MONETARY=de_DE.UTF-8    LC_MESSAGES=de_DE.UTF-8
##  [7] LC_PAPER=de_DE.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
```

```
##  [1] org.Hs.eg.db_3.14.0
##  [2] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [3] beeswarm_0.4.0
##  [4] OUTRIDER_1.99.0
##  [5] data.table_1.14.2
##  [6] SummarizedExperiment_1.24.0
##  [7] MatrixGenerics_1.6.0
##  [8] matrixStats_0.61.0
##  [9] GenomicFeatures_1.46.5
## [10] AnnotationDbi_1.56.2
## [11] Biobase_2.54.0
## [12] GenomicRanges_1.46.1
## [13] GenomeInfoDb_1.30.1
## [14] IRanges_2.28.0
## [15] S4Vectors_0.32.4
## [16] BiocGenerics_0.40.0
## [17] BiocParallel_1.28.3
## [18] knitr_1.37
##
## loaded via a namespace (and not attached):
##   [1] backports_1.4.1          BiocFileCache_2.2.1
##   [3] plyr_1.8.6               lazyeval_0.2.2
##   [5] splines_4.1.2            usethis_2.1.5
##   [7] ggplot2_3.3.5            digest_0.6.29
##   [9] foreach_1.5.2            htmltools_0.5.2
##  [11] viridis_0.6.2            fansi_1.0.2
##  [13] magrittr_2.0.2           checkmate_2.0.0
##  [15] memoise_2.0.1            BBmisc_1.12
##  [17] remotes_2.4.2            Biostrings_2.62.0
##  [19] annotate_1.72.0          prettyunits_1.1.1
##  [21] colorspace_2.0-3         blob_1.2.2
##  [23] rappdirs_0.3.3           ggrepel_0.9.1
##  [25] xfun_0.30                dplyr_1.0.8
##  [27] callr_3.7.0              crayon_1.5.0
##  [29] RCurl_1.98-1.6           jsonlite_1.8.0
##  [31] genefilter_1.76.0        iterators_1.0.14
##  [33] survival_3.3-1           glue_1.6.2
##  [35] registry_0.5-1           gtable_0.3.0
##  [37] zlibbioc_1.40.0          XVector_0.34.0
##  [39] webshot_0.5.2            DelayedArray_0.20.0
##  [41] pkgbuild_1.3.1           scales_1.1.1
##  [43] pheatmap_1.0.12          DBI_1.1.2
##  [45] Rcpp_1.0.8.2             viridisLite_0.4.0
##  [47] xtable_1.8-4             progress_1.2.2
##  [49] reticulate_1.24          bit_4.0.4
```

```
##  [51] htmlwidgets_1.5.4        httr_1.4.2
##  [53] dir.expiry_1.2.0         RColorBrewer_1.1-2
##  [55] ellipsis_0.3.2           farver_2.1.0
##  [57] pkgconfig_2.0.3          XML_3.99-0.9
##  [59] dbplyr_2.1.1             here_1.0.1
##  [61] locfit_1.5-9.5           utf8_1.2.2
##  [63] labeling_0.4.2           reshape2_1.4.4
##  [65] tidyselect_1.1.2         rlang_1.0.2
##  [67] PRROC_1.3.1              munsell_0.5.0
##  [69] tools_4.1.2              cachem_1.0.6
##  [71] cli_3.2.0                generics_0.1.2
##  [73] RSQLite_2.2.11           devtools_2.4.3
##  [75] evaluate_0.15            stringr_1.4.0
##  [77] fastmap_1.1.0            heatmaply_1.3.0
##  [79] yaml_2.3.5               processx_3.5.2
##  [81] bit64_4.0.5              fs_1.5.2
##  [83] purrr_0.3.4              KEGGREST_1.34.0
##  [85] dendextend_1.15.2        nlme_3.1-155
##  [87] xml2_1.3.3               biomaRt_2.50.3
##  [89] BiocStyle_2.22.0         brio_1.1.3
##  [91] compiler_4.1.2           plotly_4.10.0
##  [93] filelock_1.0.2           curl_4.3.2
##  [95] png_0.1-7                testthat_3.1.2
##  [97] tibble_3.1.6             geneplotter_1.72.0
##  [99] stringi_1.7.6            highr_0.9
## [101] basilisk.utils_1.6.0     ps_1.6.0
## [103] desc_1.4.1               lattice_0.20-45
## [105] Matrix_1.4-0             vctrs_0.3.8
## [107] pillar_1.7.0             lifecycle_1.0.1
## [109] BiocManager_1.30.16      bitops_1.0-7
## [111] seriation_1.3.5          rtracklayer_1.54.0
## [113] pcaMethods_1.86.0        R6_2.5.1
## [115] BiocIO_1.4.0             TSP_1.2-0
## [117] gridExtra_2.3            sessioninfo_1.2.2
## [119] codetools_0.2-18         assertthat_0.2.1
## [121] pkgload_1.2.4            DESeq2_1.34.0
## [123] rprojroot_2.0.2          rjson_0.2.21
## [125] withr_2.5.0              GenomicAlignments_1.30.0
## [127] Rsamtools_2.10.0         GenomeInfoDbData_1.2.7
## [129] mgcv_1.8-39              parallel_4.1.2
## [131] hms_1.1.1                grid_4.1.2
## [133] tidyr_1.2.0              basilisk_1.6.0
## [135] rmarkdown_2.13           restfulr_0.0.13
```