OUTRIDER - OUTlier RNA-Seq fInDER

***Felix Brechtmann*[1], *Christian Mertes*[1], *Agne Matuseviciute*[1], *Vicente Yepez*[1], *Julien Gagneur*[1]**

[1] Technische Universität München, Department of Informatics, Garching, Germany

March 28, 2018

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 which 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 additionaly provide an autoencoder, which automatically controls for covariation before fitting. After the fit, each sample can be tested for aberrantly expressed genes. Furthermore, *OUTRIDER* provides methods to easily filter unexpressed genes and visualize the results.

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

Brechtmann F, Matuseviciute A, Mertes C, Yepez V, Avsec Z, Bader D, Prokisch H, Gagneur J, *et al.*
Detection of aberrant expression - OUTlier Rna-Seq fInDER
bioRxiv

Contents

1 Introduction

OUTRIDER (OUTlier in Rna-seq fInDER) is a tool to find aberrantly expressed genes in RNA-seq samples. It does so by fitting a negative binomial model on RNA-seq read counts, correcting for variatiosn in sequeiuncg depth and appranet co-variations across samples. Read counts significantly deviating from the distribution are detected as outliers.

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

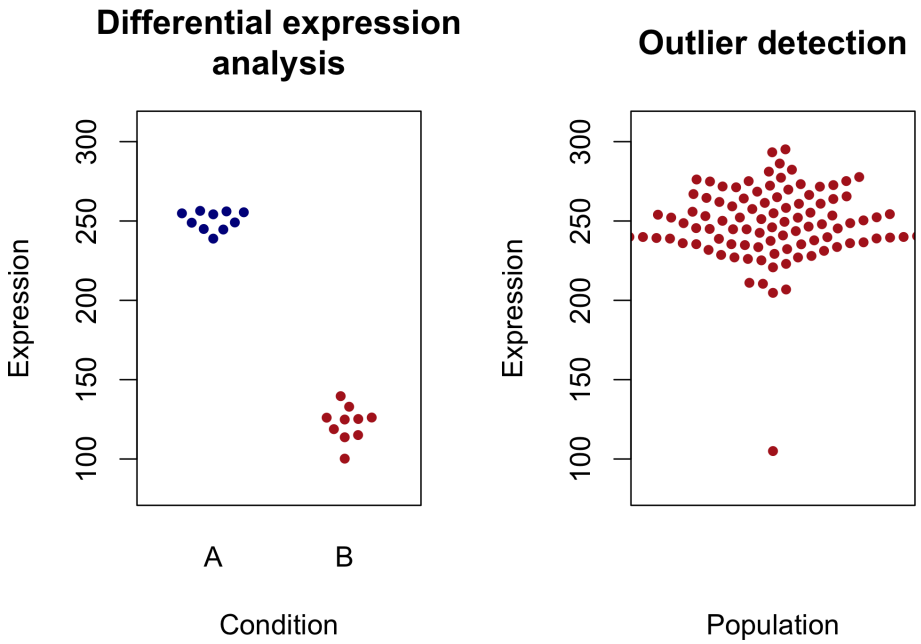


Figure 1: Scheme of workflow differences
Differences between differential gene expression analysis and outlier detection.

2 Prerequisites

To get started on the preprocessing, step we recommend to read the introductions of the aforementioned tools or the RNA-seq workflow from Bioconductor. In brief, one starts usually of with the raw FASTQ files from the sequencing run. Those are then aligned to a given reference genome. As of now (March 2018) we recommend

the STAR aligner. After obtaining the aligned BAM files one can map the reads to exons of a GTF annotation file using HT-seq. The resulting count table can then be loaded into the *OUTRIDER* package as we will describe below.

2.1 Using virtual environments

When using virtual environments in python to manage packages, we need to tell it our R session. To do so, run the following code before you start an *OUTRIDER* analysis. We assume that the pythen environment was installed into `/python-env-OUTRIDER`.

```
# check if the given python environment exists
pyEnvDir <- "~/python-env-OUTRIDER"
if(file.exists(file.path(pyEnvDir, 'bin/activate'))){
    library(reticulate)
    use_virtualenv(pyEnvDir)
}
```

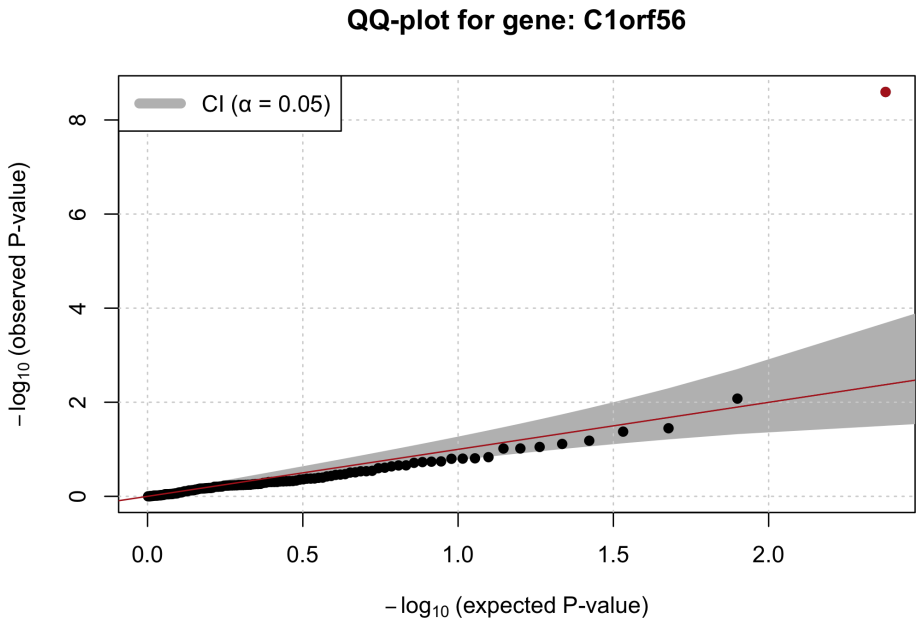
3 A quick tour

Here we assume that we already have a count table and no additional preprocessing needs to be done. Then we can start and obtain results with 3 commands. First, create an *OutriderDataSet* from a count table, then run the full pipeline using the command `OUTRIDER`. In a third step, a results table can be generated from the *OutriderDataSet* with the `results` function. Furthermore, analysis plots, which are described in section 5, can be made 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, onlyZeros=TRUE, filterGenes=TRUE)

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

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

```
##      geneID sampleID       pValue      padjust zScore l2fc rawcounts
## 1:  C1orf56    74123 2.538935e-09 1.987986e-06   4.63 2.52       220
## 2: CYB561D1    74123 7.769843e-09 3.041893e-06   4.08 3.16       686
## 3:   SETDB1    74123 3.872592e-08 1.010746e-05   4.13 2.48       995
## 4:   VANGL1  MUC1379 2.499240e-08 1.956905e-05   3.37 3.94      2160
## 5:   RNF207    74123 1.071628e-07 2.097713e-05   4.01 2.76       444
## 6: AK307246    74123 3.844084e-07 6.019836e-05   3.74 2.67       257
##    normcounts   mu disp meanCorrected AberrantBySample AberrantByGene
## 1:     639.38 1.03 6.18     110.43168               55              1
## 2:    8407.03 1.09 2.79     937.97856               55              1
## 3:    6726.34 1.04 5.14    1206.27335               55              1
## 4:   63612.42 1.04 1.25    4154.38977                1              1
## 5:    1019.74 1.07 3.57     150.25358               55              1
## 6:     587.46 1.00 3.59      91.13074               55              1
##    padj_rank
## 1:         1
## 2:         2
## 3:         3
## 4:         1
## 5:         4
## 6:         5
```

```
#example of a QQplot for the first outlier.
geneID <- res[1, geneID]
plotQQ(ods, geneID)
```



4 *OUTRIDER* analysis in detail

For this Tutorial we will use the full rare disease data set from Kremer and Bader et al. For testing this package contains also a subset of this.

```
URL <- paste0("https://media.nature.com/original/nature-assets/",
        "ncomms/2017/170612/ncomms15824/extref/ncomms15824-s1.txt")
ctsTable <- read.table(URL, sep="\t")

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

4.1 Preprocessing

It is recommendable to apply some data preprocessing before fitting. Our model requires that there is at least one count in each gene, and for large data sets at least one count in 100 samples. Therefore, all genes that are not expressed at all must be discarded.

In a specific tissue not all genes are expressed. Therefore, we provide the function `filterExpression` to remove genes which have a low FPKM expression value. 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.

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 into 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"))
```

The `TxDb.Hsapiens.UCSC.hg19.knownGene` only contains well annotated genes. This annotation will miss a lot of genes. 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].

```
library(RMySQL)
library(AnnotationDbi)
txdbUrl <- paste0("https://i12g-gagneurweb.in.tum.de/public/",
        "paper/mitoMultiOmics/ucsc.knownGenes.db")
```

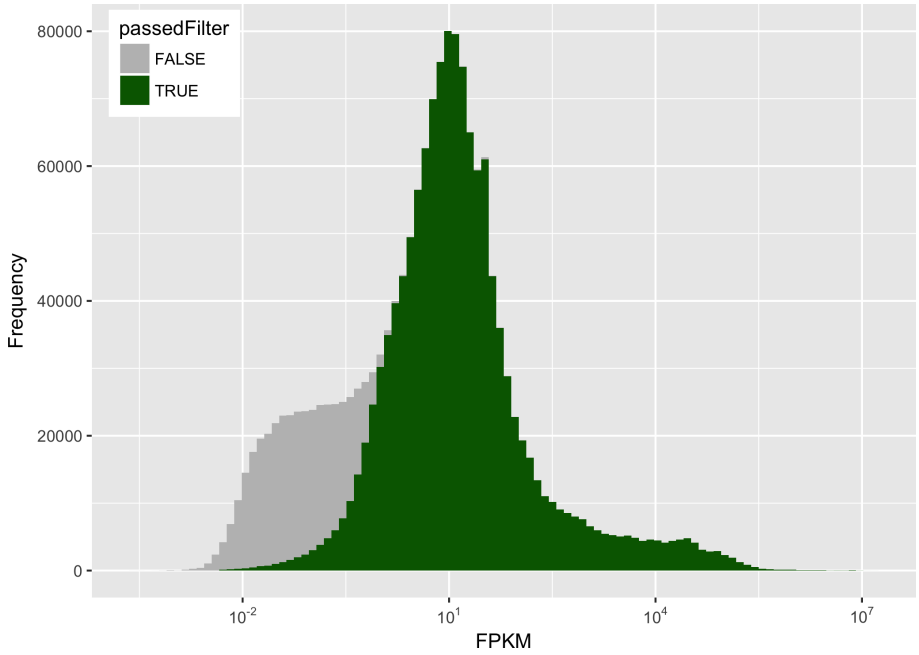
[1] https://i12g-gagneurweb.in.tum.de/public/paper/mitoMultiOmics/ucsc.knownGenes.db

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

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

```
download.file(txdbUrl, "ucsc.knownGenes.db")
txdb <- loadDb("ucsc.knownGenes.db")
con <- dbConnect(MySQL(), host='genome-mysql.cse.ucsc.edu',
        dbname="hg19", user='genome')
map <- dbGetQuery(con, 'select kgId AS TXNAME, geneSymbol from kgXref')
```

```
# Calculating fpkm values and only labeling not expressed genes
ods <- filterExpression(ods, txdb, mapping=map,
        filterGenes=FALSE, savefpkm=TRUE)

# Display the pre distribution of counts.
plotFPKM(ods)
```

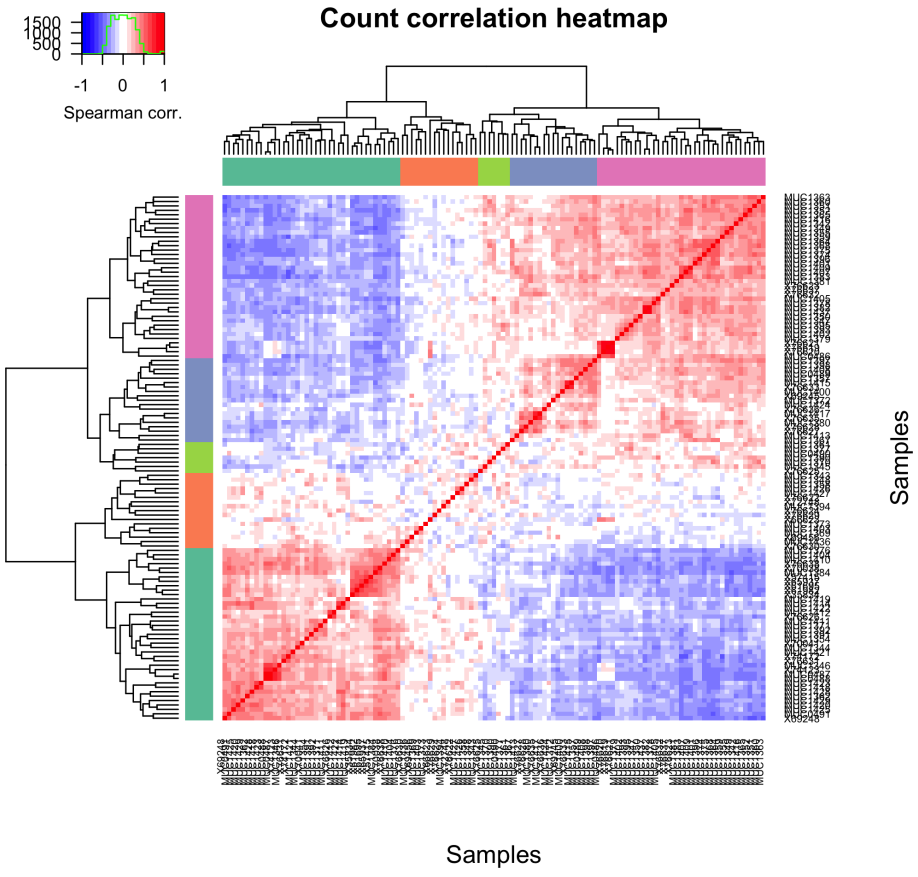



```
# do the actuall subsetting based on the filtering labels
ods <- filterExpression(ods, txdb, mapping=map,
        filterGenes=TRUE, savefpkm=TRUE)
```

4.2 Controlling for Confounders

A next step in any analysis workflow is to visualize the correlations between samples. Here, we observe that samples are correlated. These correlations are often due to confounders, technical like the sequencing batch, or biological ones like gender. These cofounders can harm the detection of aberrant features. Therefore, we provide options to control for them.

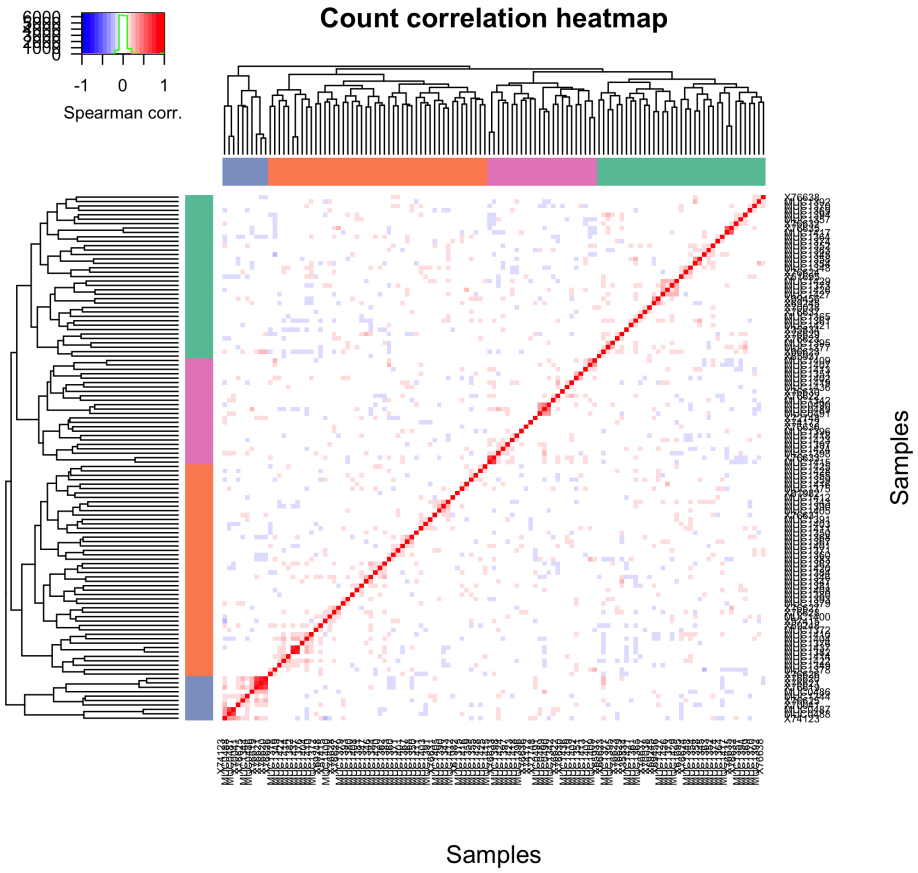
```
# Heatmap plot sample correlation
# It annotates also the clusters as of the dendrogram
ods <- plotCountCorHeatmap(ods, normalized=FALSE, nCluster=5)
```



Now 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*.

Additionally calling the `autoCorrect` function calls an autoencoder, which automatically controls for confounders present in the data. After controlling for cofounders the heatmap should be plotted again. If it worked, no batches should be present and the correlations between samples should be close to zero.

```
# Control for confounders
ods <- estimateSizeFactors(ods)
ods <- autoCorrect(ods)

# Heatmap plot sample correlation
ods <- plotCountCorHeatmap(ods, normalized=TRUE)
```



Alternatively, a `normalizationFactor` matrix can be provided. It must be computed beforehand using any method. Its purpose is to normalize for technical effects or control for additional expression patterns.

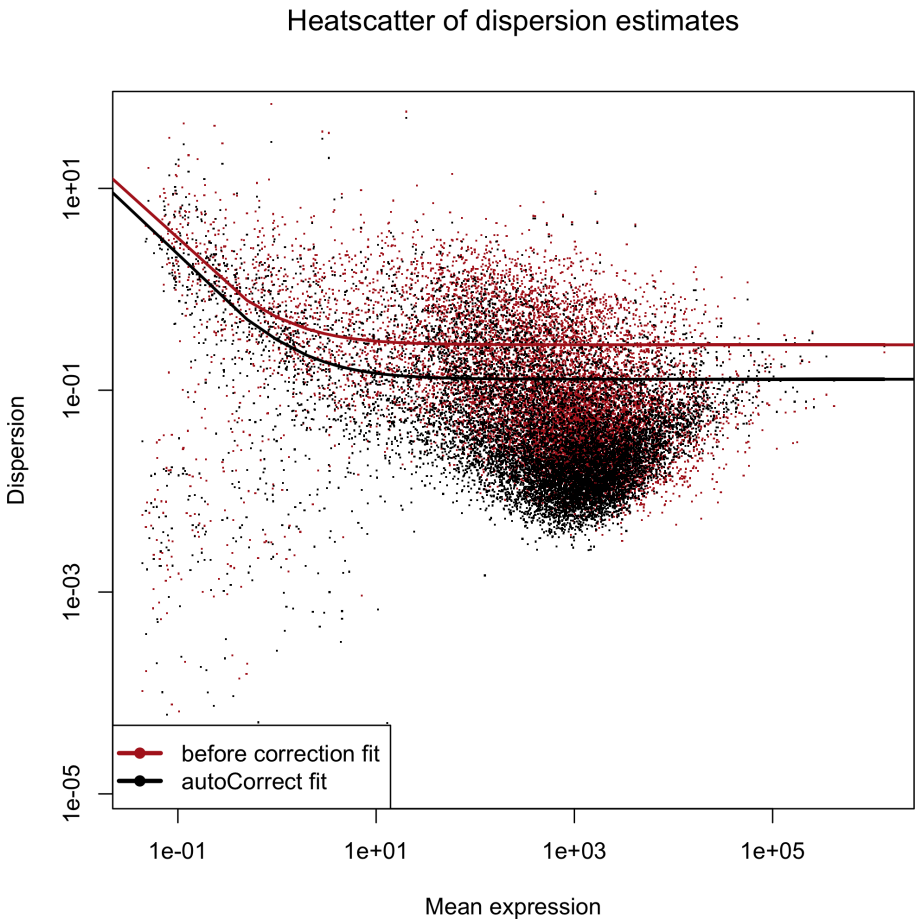
4.3 Fitting

After all the preprocessing part, we can finally start the fitting and testing. We provide a single wrapper function `OUTRIDER` to run the full pipeline, but it can also be run step by step. By default the `OUTRIDER` function runs all analysis functions and controlls the counts using autoCorrect. If autoCorrect is not installed, a warning is returned and the analysis skips the correction part automatically. It does not include any preprocessing functions. If counts need to be discared it has to be done manually before running the `OUTRIDER` function or starting the analysis pipeline.

To fit the dispersion and the mean, the `fit` function is applied to the *OutriderDataSet*.

```
# fit NB to feature counts
ods <- fit(ods)

# plot dispersion versus mean counts
```

```
plotDispEsts(ods)
```



4.4 P-value computation

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

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

where the $\frac{1}{2}$ term handles the case of both terms exceeding 0.5, which can happen due to the discrete nature of the negative binomial distribution. If required a one-sided test can be computed using the alternative argument 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-HochbergâĂŹs False Discovery Rate method (Benjamini and Hochberg 1995). Alternatively all other by `p.adjust` supported Methods can be used via the parameter `method`.

```
# compute pValues (nominal and adjusted)
ods <- computePvalues(ods, alternative="two.sided")
```

4.5 Z-score calculation

The Z-scores on the log2 transformed counts can be used for visualization, filtering, and ranking of samples. By running the `computeZscores` function the Z-scores are precomputed and stored in the *OutriderDataSet* object. Where the Z-score is calculated as:

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

$$l_{ij} = \log\left(\frac{k_{ij}+1}{c_{ij}}\right)$$

Where μ_j^l the mean of l_{ij} for gene j and σ_j^l the standard deviation of l_{ij} for gene j is.

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

5 Results

The *OUTRIDER* package offers mutiple ways to display the results. It creates a result table containing all the values computed during the analysis. Alternatively, it offers various plot functions, which guide through the analysis.

5.1 Result table

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

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

```
##     geneID sampleID       pValue      padjust zScore  l2fc rawcounts
## 1:  NUDT12   X69456 7.469660e-21 8.625963e-17 -10.27 -8.49         0
## 2:   ATP5I  MUC1372 3.006906e-18 3.472375e-14  -9.41 -3.13        84
## 3:  NDUFB2  MUC1372 8.742570e-17 5.047960e-13  -8.88 -1.85       313
```

```
## 4:  NDUFA1  MUC1372 5.208618e-16 2.004971e-12  -8.82 -2.33       178
## 5:  COX6B1  MUC1372 3.068662e-15 8.859228e-12  -8.66 -2.34       504
## 6: NDUFB10  MUC1372 1.432280e-14 3.307994e-11  -8.21 -1.35       365
##    normcounts   mu   disp meanCorrected AberrantBySample AberrantByGene
## 1:       0.00 1.07  14.49      470.5967               4              1
## 2:     155.38 1.04  33.27     1379.6981             800              1
## 3:     493.93 1.04  68.75     1785.1664             800              1
## 4:     295.19 1.00  44.81     1491.3877             800              1
## 5:     484.79 1.02  39.60     2464.3263             800              1
## 6:     533.13 1.02 105.37     1362.8954             800              1
##    padj_rank
## 1:         1
## 2:         1
## 3:         2
## 4:         3
## 5:         4
## 6:         5
```

```
# setting a different significant level
res <- results(ods, padj=0.1)
head(res)
```

```
##     geneID sampleID       pValue      padjust zScore  l2fc rawcounts
## 1:  NUDT12   X69456 7.469660e-21 8.625963e-17 -10.27 -8.49         0
## 2:   ATP5I  MUC1372 3.006906e-18 3.472375e-14  -9.41 -3.13        84
## 3:  NDUFB2  MUC1372 8.742570e-17 5.047960e-13  -8.88 -1.85       313
## 4:  NDUFA1  MUC1372 5.208618e-16 2.004971e-12  -8.82 -2.33       178
## 5:  COX6B1  MUC1372 3.068662e-15 8.859228e-12  -8.66 -2.34       504
## 6: NDUFB10  MUC1372 1.432280e-14 3.307994e-11  -8.21 -1.35       365
##    normcounts   mu   disp meanCorrected AberrantBySample AberrantByGene
## 1:       0.00 1.07  14.49      470.5967               4              1
## 2:     155.38 1.04  33.27     1379.6981             800              1
## 3:     493.93 1.04  68.75     1785.1664             800              1
## 4:     295.19 1.00  44.81     1491.3877             800              1
## 5:     484.79 1.02  39.60     2464.3263             800              1
## 6:     533.13 1.02 105.37     1362.8954             800              1
##    padj_rank
## 1:         1
## 2:         1
## 3:         2
## 4:         3
## 5:         4
## 6:         5
```

5.2 Number of aberrant genes per sample

One quantity of interest is the number of aberrantly expressed genes per sample. This can be diplayed using the `plotAberrantPerSample` plotting function. Alternatively, the `aberrant` function can be used to compute the number of aberrant counts. Those can be computed by sample, gene or in the whole data set. These numbers depend on the cutoffs, which can be specified in both functions.

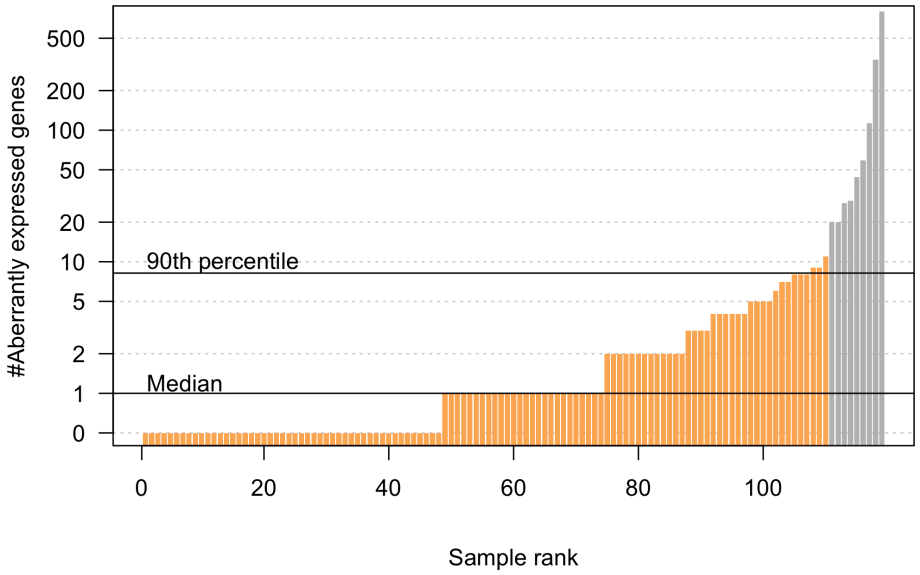
```
# Number of aberrant genes per sample
tail(sort(aberrant(ods, by="sample")))
```

```
## MUC1403 MUC0489  X76626 MUC1421  X74123 MUC1372
##      29      44      59     113     344     800
```

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

```
##         TST      UTP14A    CXorf40A       MXRA5     TIMM17B SLMO2-ATP5E
##           2           2           2           2           2           3
```

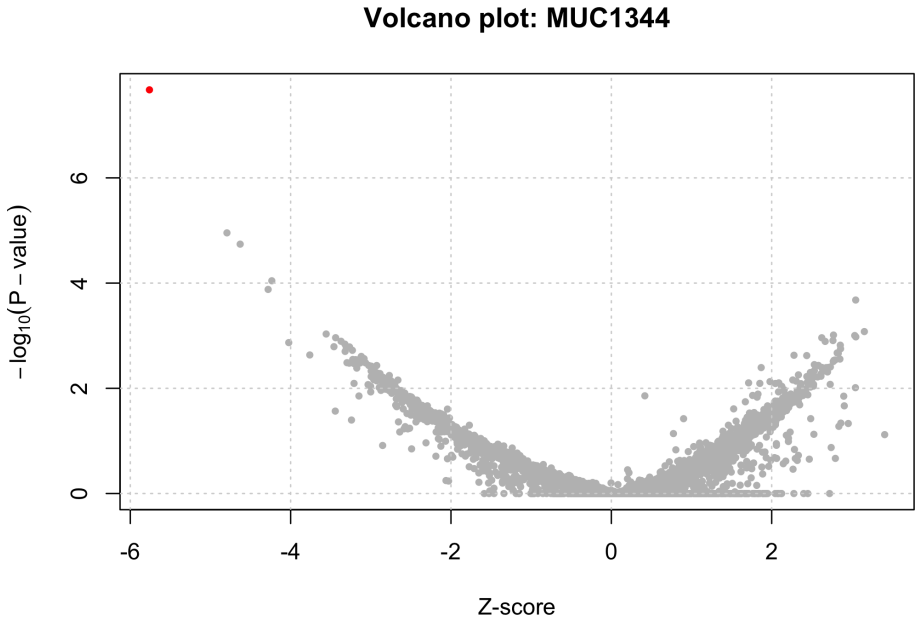
```
# Plotting the aberrant events per sample
plotAberrantPerSample(ods, padj=0.05)
```



5.3 Volcano plots

To view the distribution of P-values on a sample level, volcano plots can be displayed. Most of the plots do make use of the *plotly* framework to create interactive plots. For the vignette we will always use the basic R functionality from *graphics*.

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

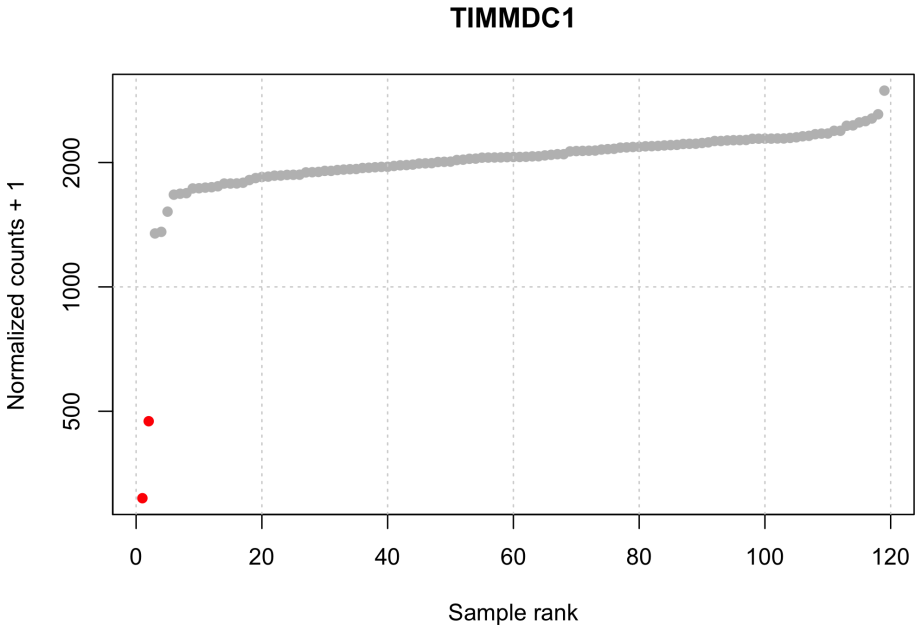


5.4 Gene plots

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

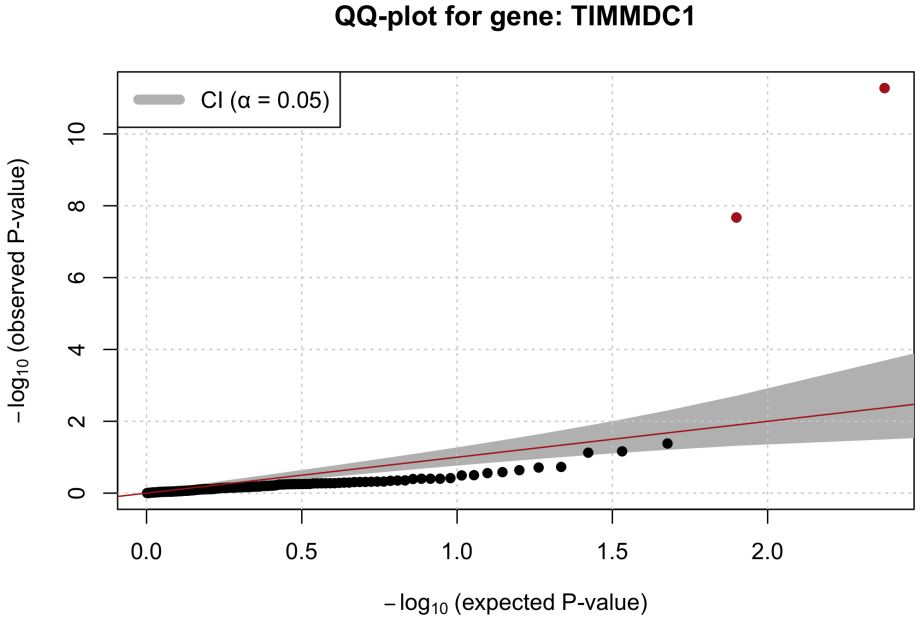
When using the *plotly* framework for plotting. all computed values are displayed for each data point. The user can access this information by hovering over each data point with the mouse.

```
# Expression rank of an outlier gene
plotExpressionRank(ods, "TIMMDC1", basePlot=TRUE)
```



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

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



6 Saving and loading of models

Fit parameters can be saved and loaded to simplify the analysis or to precompute models, for later use in another data set.

Eather a file path can be added in the `fit` function or the `writeNBModel` function can be used to save the model after fitting.

```
# Make an OutriderDataSet and compute the fit.
ods <- makeExampleOutriderDataSet()
ods <- estimateSizeFactors(ods)
ods <- fit(ods)

# Save the model to disk: 'model.tsv'.
writeNBModel(ods, 'model.tsv')

## NULL
```

After saving the model to disk the model can be used in a new analysis where we want to reuse it without fitting again the data. In the following code chunk we show how we load the model from disk into a newly created *OutriderDataSet* object. Afterwards, `estimateSizeFactors` has to be called to compute size factors, wich are based on the loaded parameter set. Then the analysis functions `computePvalues` and `computeZscores` can be called.

```
# Create a new data set and load the model
ods <- makeExampleOutriderDataSet()
ods <- readNBModel(ods, 'model.tsv')

# do the analysis based on the loaded model
ods <- estimateSizeFactors(ods)
ods <- computePvalues(ods)
ods <- computeZscores(ods)
```

Alternatively the model file path can be specified in the `computePvalues` function. Then the model is loaded, the size Factors get estiamted and the P-values are computed.

In the same fashion, the autoencoder parameters can be written to or loaded from a directory. For saving, a model name and directory have to be specified in the `autoCorrect` function call. If the modell should be reused the user has to specify the same model name and directory in the `autoCorrect` function and additionally set the `predict` option to TRUE. This option can only be used in combination with the above mentioned method, to have a common reference for the size factors.

```
# Saving
ods <- makeExampleOutriderDataSet(m=50)
```

```
ods <- estimateSizeFactors(ods)
ods <- autoCorrect(ods, save=TRUE, modelName='AEModel',
        modelDirectory='.')

# Loading
ods <- makeExampleOutriderDataSet(m=100)
ods <- estimateSizeFactors(ods)
ods <- autoCorrect(ods, predict=TRUE, modelName='AEModel',
        modelDirectory='.')
```

Session info

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

```
## R version 3.4.2 (2017-09-28)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS High Sierra 10.13.3
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] org.Hs.eg.db_3.5.0
##  [2] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [3] beeswarm_0.2.3
##  [4] OUTRIDER_0.99.3
##  [5] data.table_1.10.4-3
##  [6] SummarizedExperiment_1.8.1
##  [7] DelayedArray_0.4.1
##  [8] matrixStats_0.53.1
##  [9] GenomicFeatures_1.30.3
## [10] AnnotationDbi_1.40.0
## [11] Biobase_2.38.0
## [12] GenomicRanges_1.30.3
```

```
## [13] GenomeInfoDb_1.14.0
## [14] IRanges_2.12.0
## [15] S4Vectors_0.16.0
## [16] BiocGenerics_0.24.0
## [17] BiocParallel_1.12.0
## [18] reticulate_1.6
## [19] knitr_1.20
##
## loaded via a namespace (and not attached):
##  [1] bitops_1.0-6             bit64_0.9-7
##  [3] RColorBrewer_1.1-2       progress_1.1.2
##  [5] httr_1.3.1               rprojroot_1.3-2
##  [7] tools_3.4.2              backports_1.1.2
##  [9] R6_2.2.2                 rpart_4.1-11
## [11] KernSmooth_2.23-15       Hmisc_4.0-3
## [13] DBI_0.8                  lazyeval_0.2.1
## [15] colorspace_1.3-2         nnet_7.3-12
## [17] gridExtra_2.3            prettyunits_1.0.2
## [19] DESeq2_1.18.1            RMySQL_0.10.14
## [21] bit_1.1-12               compiler_3.4.2
## [23] htmlTable_1.9            plotly_4.7.1
## [25] labeling_0.3             rtracklayer_1.38.3
## [27] caTools_1.17.1           scales_0.5.0
## [29] checkmate_1.8.5          genefilter_1.60.0
## [31] stringr_1.3.0            digest_0.6.15
## [33] Rsamtools_1.30.0         foreign_0.8-69
## [35] rmarkdown_1.9            XVector_0.18.0
## [37] base64enc_0.1-3          pkgconfig_2.0.1
## [39] htmltools_0.3.6          highr_0.6
## [41] htmlwidgets_0.9          rlang_0.2.0
## [43] RSQLite_2.0              BBmisc_1.11
## [45] bindr_0.1                jsonlite_1.5
## [47] gtools_3.5.0             acepack_1.4.1
## [49] dplyr_0.7.4              RCurl_1.95-4.10
## [51] magrittr_1.5             GenomeInfoDbData_0.99.1
## [53] Formula_1.2-2            Matrix_1.2-12
## [55] Rcpp_0.12.16             munsell_0.4.3
## [57] stringi_1.1.6            yaml_2.1.18
## [59] zlibbioc_1.24.0          gplots_3.0.1
## [61] plyr_1.8.4               grid_3.4.2
## [63] blob_1.1.0               gdata_2.18.0
## [65] lattice_0.20-35          Biostrings_2.46.0
## [67] splines_3.4.2            annotate_1.56.1
## [69] locfit_1.5-9.1           pillar_1.2.1
## [71] reshape2_1.4.2           codetools_0.2-15
```

```
## [73] geneplotter_1.56.0       biomaRt_2.34.2
## [75] XML_3.98-1.10            glue_1.2.0
## [77] evaluate_0.10.1          latticeExtra_0.6-28
## [79] tidyr_0.8.0              purrr_0.2.4
## [81] gtable_0.2.0             assertthat_0.2.0
## [83] ggplot2_2.2.1            xtable_1.8-2
## [85] viridisLite_0.2.0        survival_2.41-3
## [87] tibble_1.4.2             GenomicAlignments_1.14.1
## [89] memoise_1.1.0            bindrcpp_0.2
## [91] cluster_2.0.6            LSD_4.0-0
## [93] BiocStyle_2.6.1
```