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

May 24, 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 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 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, Herzog M, Bader D M, Prokisch H, Gagneur J; **OUTRIDER: A statistical method for detecting aberrantly expressed genes in RNA sequencing data**; *bioRxiv*; 2018

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 to RNA-seq read counts, correcting for variations in sequencing depth and apparent 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. 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 between groups of samples. In contrast, *OUTRIDER* aims at detecting outliers within a given population. A scheme of this difference is given in figure 1.

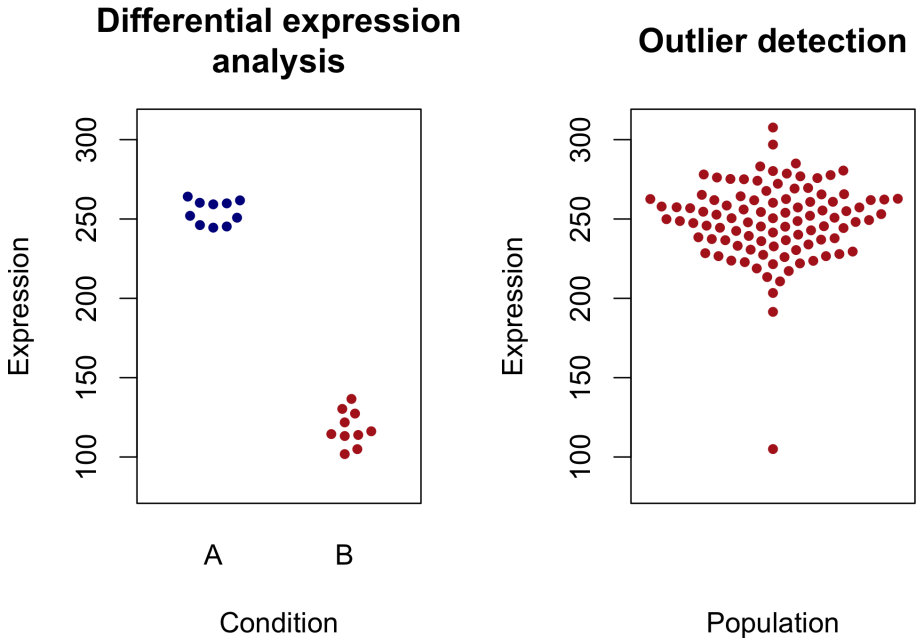


Figure 1: Scheme of workflow differences
Differences between differential gene expression analysis and outlier detection.

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 usually starts with the raw FASTQ files from the RNA sequencing run. Those are then aligned to a given reference genome. As of now (May 2018), we recommend

the STAR aligner. After obtaining the aligned BAM files, one can map the reads to exons or genes 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.

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 created from the *OutriderDataSet* object directly.

```
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)
```

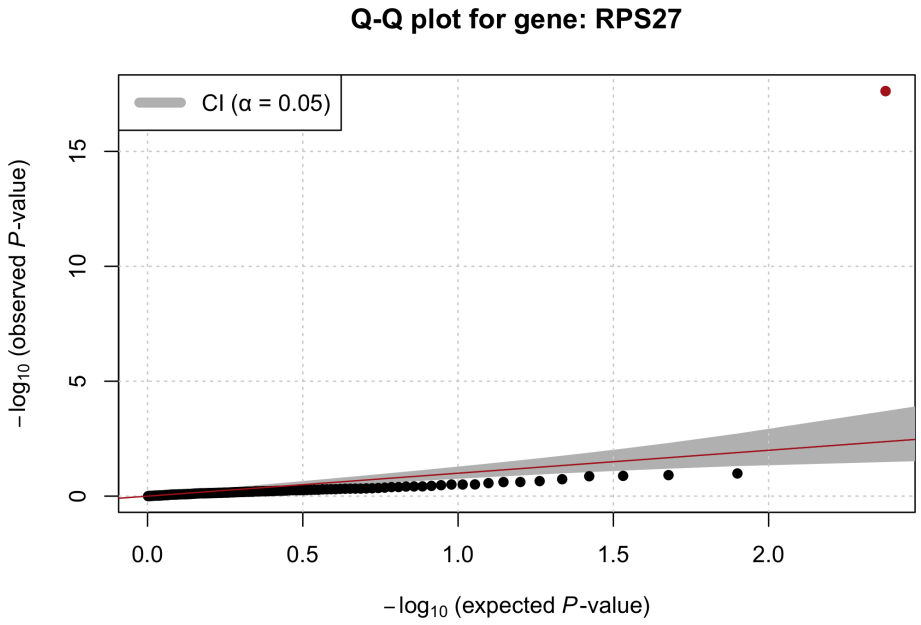
```
## [1] "Initial PCA loss: 4.62018005328588"
## [1] "Time elapsed: 5.2010509967804"
## [1] "nb-PCA loss: 4.43612213768131"
```

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

```
##     geneID sampleID       pValue      padjust zScore  l2fc rawcounts
## 1:   RPS27  MUC1372 2.381861e-18 1.350442e-14  -9.33 -2.29       949
## 2:   PARK7  MUC1372 1.522134e-15 4.315017e-12  -8.31 -0.97      1354
## 3:   UQCRH  MUC1372 6.796505e-15 1.284470e-11  -8.35 -1.54       490
## 4:   RPL11  MUC1372 9.449677e-15 1.339419e-11  -8.29 -1.52      2713
## 5:  MRPS21  MUC1372 4.120122e-13 4.671966e-10  -7.81 -1.29       300
## 6: LAMTOR2  MUC1372 5.677835e-13 5.365263e-10  -7.82 -1.26       185
##    normcounts mu   disp meanCorrected AberrantBySample AberrantByGene
## 1:    2241.49  1  49.75     10978.672               36              1
## 2:    2295.90  1 191.74      4494.130               36              1
```

```
## 3:     905.71  1  80.50      2643.330               36             1
## 4:    5437.72  1  76.48     15599.891               36             1
## 5:     607.05  1 104.15      1492.087               36             1
## 6:     351.13  1 124.64       844.167               36             1
##    padj_rank
## 1:         1
## 2:         2
## 3:         3
## 4:         4
## 5:         5
## 6:         6
```

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



4 An *OUTRIDER* analysis in detail

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

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

We provide the function `filterExpression` to remove genes which have low FPKM 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.

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"))
```

The `TxDb.Hsapiens.UCSC.hg19.knownGene` contains only 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")
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')
```

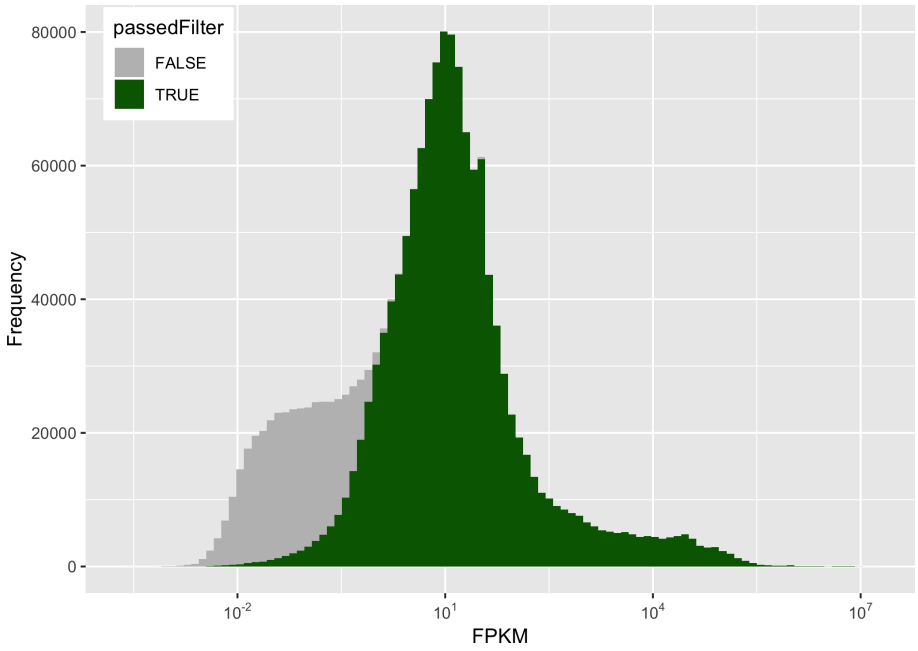
```
# calculating FPKM values and only labeling not expressed genes
ods <- filterExpression(ods, txdb, mapping=map,
        filterGenes=FALSE, savefpkm=TRUE)
```

[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

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

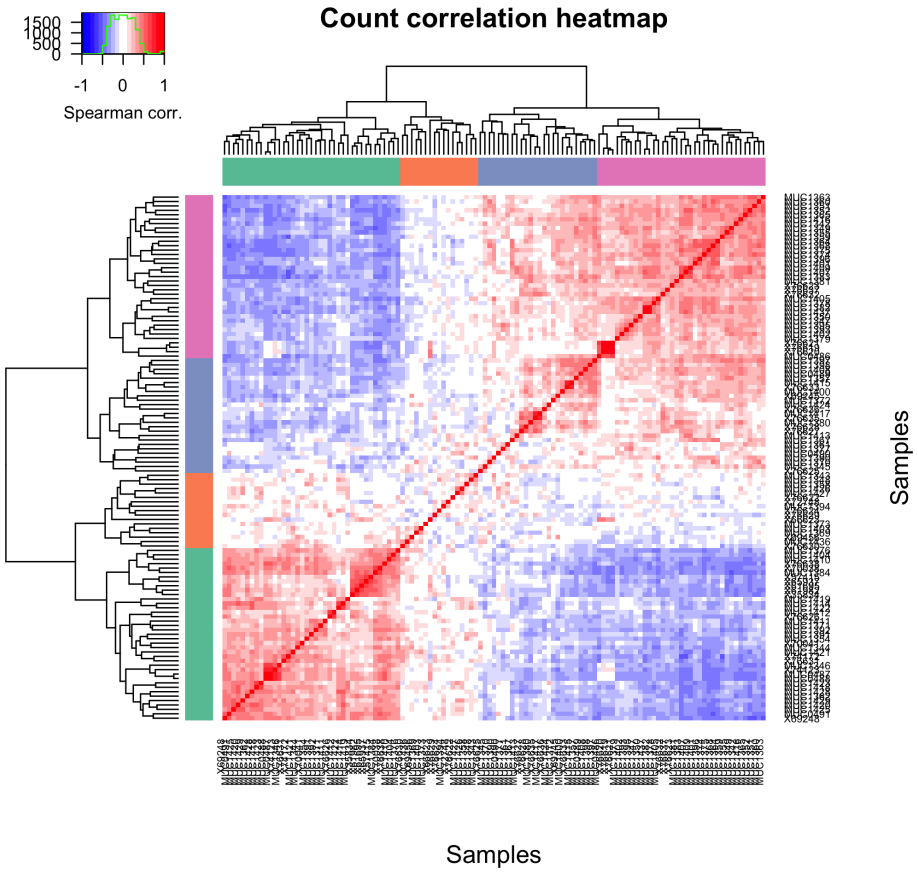



```
# do the actual subsetting based on the filtering labels
ods <- filterExpression(ods, txdb, mapping=map,
        filterGenes=TRUE, savefpkm=TRUE)
```

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 sex. These confounders can harm the detection of aberrant features. Therefore, we provide options to control for them.

```
# Heatmap of the sample correlation
# it annotates also the clusters resulting from the dendrogram
ods <- plotCountCorHeatmap(ods, normalized=FALSE, nCluster=4)
```



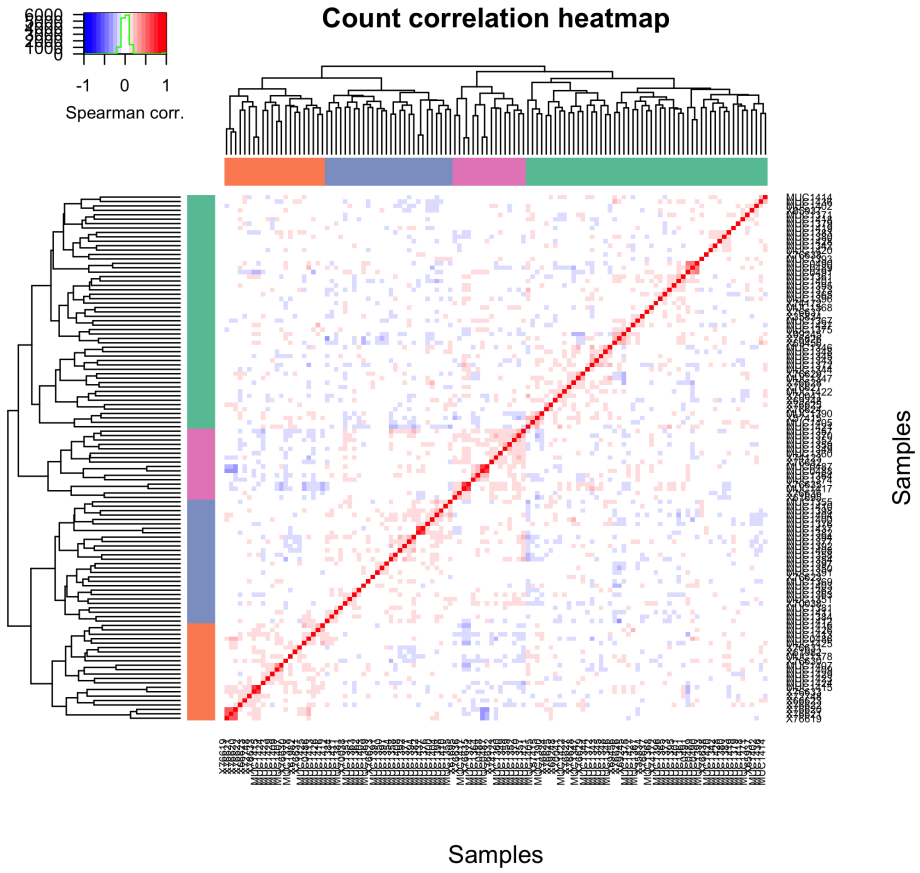
We have different ways to control for confounders present in the data. The first and standard way is to calculate the `sizeFactors` as done in *DESeq2*.

Additionally, the `autoCorrect` function calls an autoencoder, which automatically controls for confounders present in the data. Therefore an encoding dimension q needs to be set or the default value 20 is used. To find the optimal value the `grid SearchEncDim` After controlling for them, the heatmap should be plotted again. If it worked, no batches should be present and the correlations between samples should be close to zero.

```
# automatically control for confounders
ods <- estimateSizeFactors(ods)
ods <- autoCorrect(ods, q=13)

## [1] "Initial PCA loss: 6.24249831269439"
## [1] "Time elapsed: 1.13855073054632"
## [1] "nb-PCA loss: 6.12284209444142"

# Heatmap of the sample correlation after controlling
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.

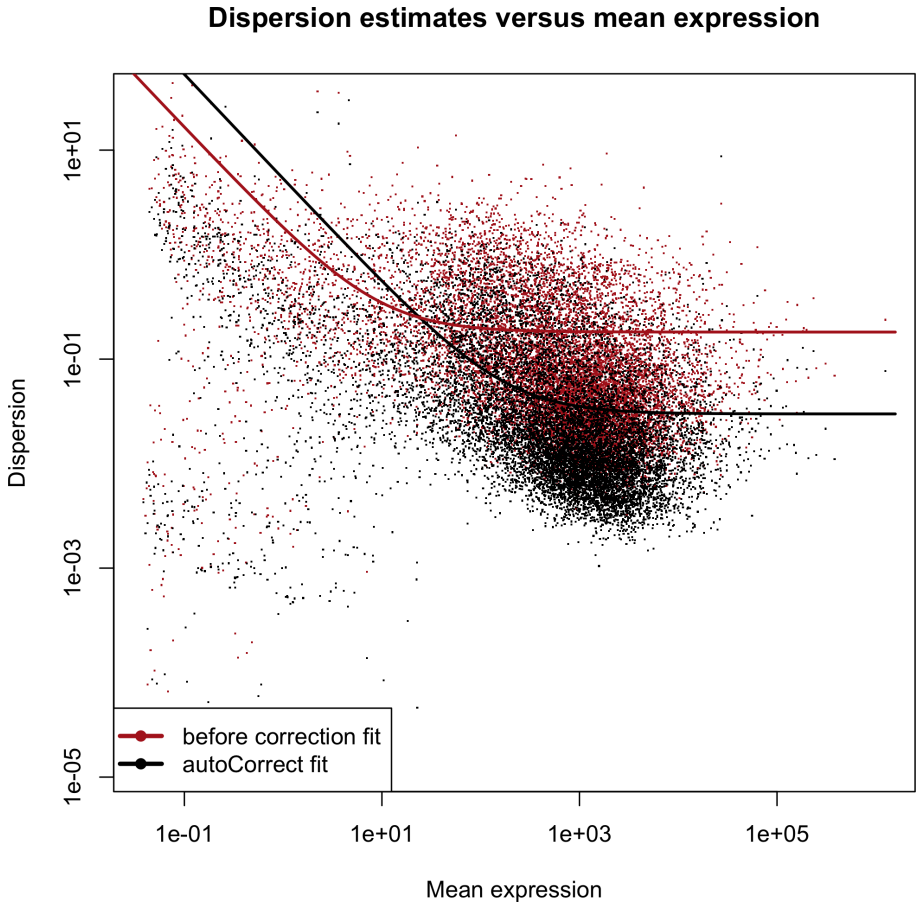
4.3 Fitting

After all the preprocessing part, we can finally start 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 controls the counts using `autoCorrect`. It does not include any preprocessing functions. Discarding genes or samples should 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 each feature
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:

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

where the $\frac{1}{2}$ term handles the case of both terms exceeding 0.5, which can happen due to the discrete nature of the negative binomial distribution. 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 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-Yekutieli's false discovery rate method (Benjamini and Yekutieli, 2001). Alternatively, all other by `p.adjust` supported methods can be used via the `method` parameter.

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

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 computed and stored in the *OutriderDataSet* object. The Z-scores are calculated using:

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

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

Where μ_j^l is the mean and σ_j^l the standard deviation of l_{ij} for gene j. Here l_{ij} are the log2 transformed counts after correction for confounders.

```
# compute and store 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, which 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 (only significant, padj < 0.05)
res <- results(ods, zScore=0)
head(res)
```

```
##    geneID sampleID       pValue      padjust zScore  l2fc rawcounts
## 1: NUDT12   X69456 4.403331e-21 5.050148e-16 -10.28 -8.52         0
## 2:  ATP5I  MUC1372 6.282652e-20 7.205527e-15  -9.73 -3.29        84
## 3:  STAG2   X76636 3.721317e-18 4.267951e-13  -9.24 -2.07       622
```

```
## 4: COX6B1  MUC1372 1.909880e-17 8.325938e-13  -9.03 -1.83       504
## 5:  RPS27  MUC1372 2.177869e-17 8.325938e-13  -9.19 -2.64       949
## 6:  COX7B  MUC1372 6.625980e-17 9.954982e-13  -8.93 -1.90       326
##    normcounts mu  disp meanCorrected AberrantBySample AberrantByGene
## 1:       0.00  1 14.56      471.6234                1              1
## 2:     139.47  1 33.94     1380.3575              866              1
## 3:     996.83  1 59.13     4179.8818                6              1
## 4:     684.46  1 70.50     2440.8304              866              1
## 5:    1758.88  1 37.41    10955.5302              866              1
## 6:     435.89  1 65.49     1634.4783              866              1
##    padj_rank
## 1:       1.0
## 2:       1.0
## 3:       1.0
## 4:       2.5
## 5:       2.5
## 6:       6.0
```

```
dim(res)
```

```
## [1] 1060   14
```

```
# setting a different significant level or filtering for Z-scores
res <- results(ods, padj=0.1, zScore=2)
head(res)
```

```
##    geneID sampleID       pValue      padjust zScore  l2fc rawcounts
## 1: NUDT12   X69456 4.403331e-21 5.050148e-16 -10.28 -8.52         0
## 2:  ATP5I  MUC1372 6.282652e-20 7.205527e-15  -9.73 -3.29        84
## 3:  STAG2   X76636 3.721317e-18 4.267951e-13  -9.24 -2.07       622
## 4: COX6B1  MUC1372 1.909880e-17 8.325938e-13  -9.03 -1.83       504
## 5:  RPS27  MUC1372 2.177869e-17 8.325938e-13  -9.19 -2.64       949
## 6:  COX7B  MUC1372 6.625980e-17 9.954982e-13  -8.93 -1.90       326
##    normcounts mu  disp meanCorrected AberrantBySample AberrantByGene
## 1:       0.00  1 14.56      471.6234                1              1
## 2:     139.47  1 33.94     1380.3575              866              1
## 3:     996.83  1 59.13     4179.8818                6              1
## 4:     684.46  1 70.50     2440.8304              866              1
## 5:    1758.88  1 37.41    10955.5302              866              1
## 6:     435.89  1 65.49     1634.4783              866              1
##    padj_rank
## 1:       1.0
## 2:       1.0
## 3:       1.0
## 4:       2.5
## 5:       2.5
## 6:       6.0
```

```
dim(res)
```

```
## [1] 1228   14
```

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 `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.

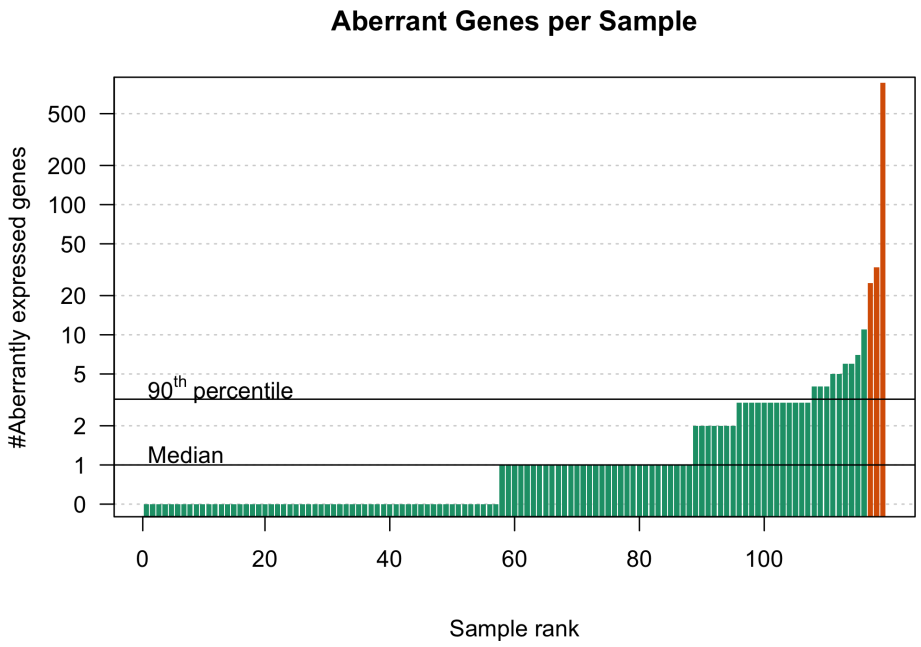
```
# number of aberrant genes per sample
tail(sort(aberrant(ods, by="sample")))
```

```
##  X76636 MUC1362  X76630 MUC1403 MUC1421 MUC1372
##       6       7      11      25      33     866
```

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

```
##        AMPH        ZFAT    ADAMTSL1      DNAJC3        ICT1 SLMO2-ATP5E
##           2           2           2           2           2           3
```

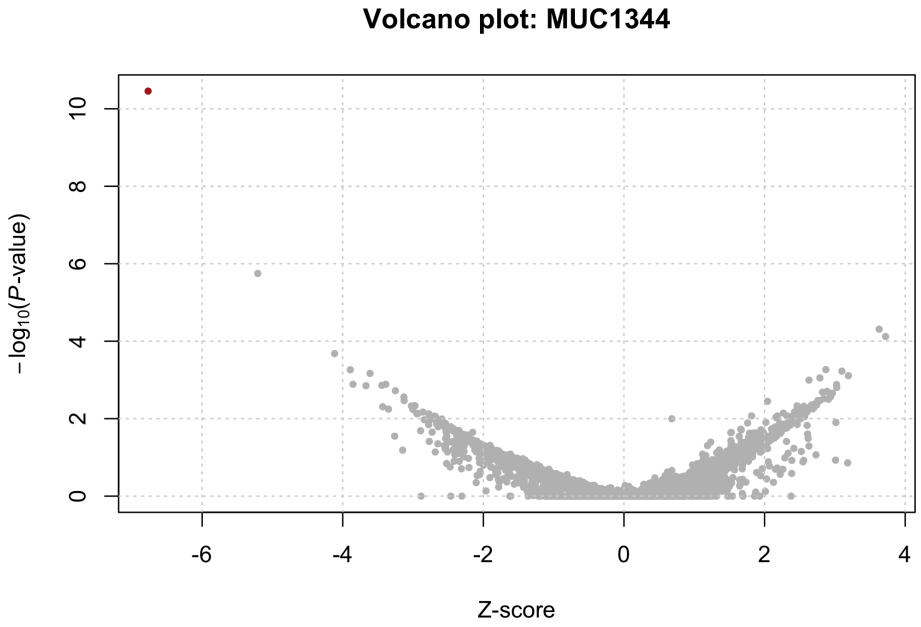
```
# plotting the aberrant events per sample
plotAberrantPerSample(ods, padj=0.05)
```



5.3 Volcano plots

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

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

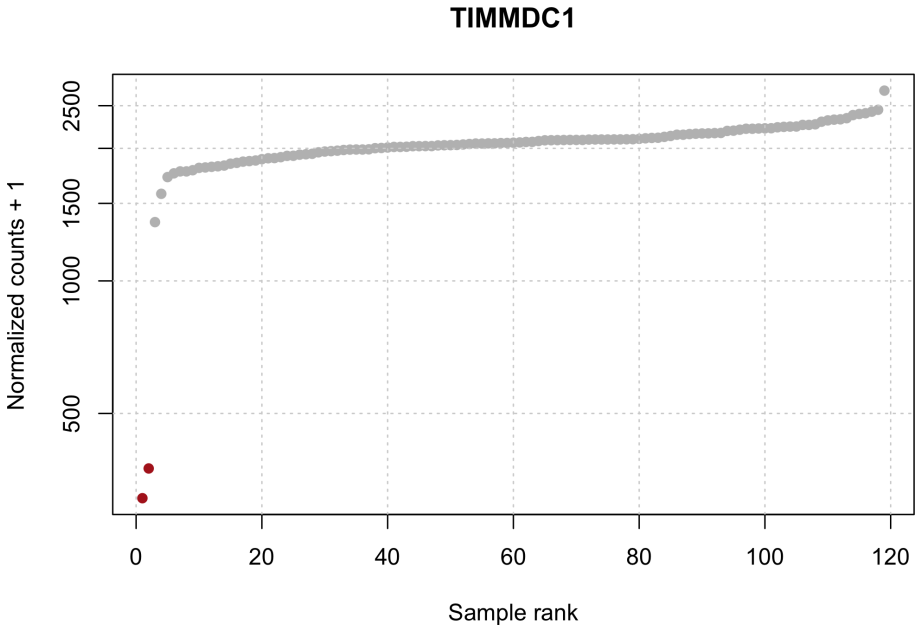


5.4 Gene plots

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.

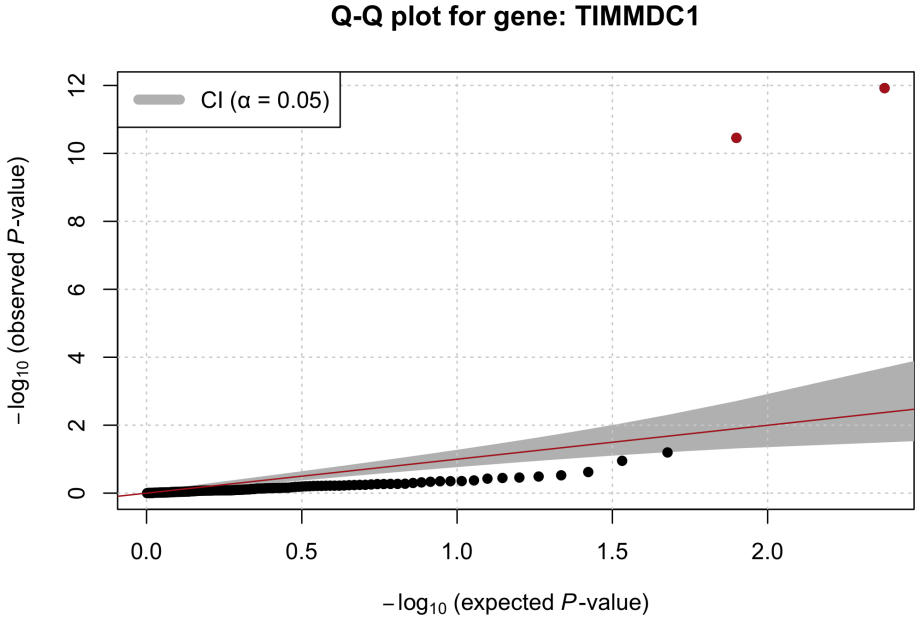
When using the *plotly* framework for plotting, all computed values are displayed for each data point. The user can access this information by hovering over each data point with the mouse.

```
# expression rank of an outlier gene
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", legendPos="topleft")
```



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 10.13.4
##
## 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_0.99.7
##  [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] 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_0.5.0.9000
## [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] pkgconfig_2.0.1         base64enc_0.1-3
## [39] htmltools_0.3.6         highr_0.6
## [41] htmlwidgets_0.9         rlang_0.2.0.9001
## [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] reticulate_1.7          stringi_1.1.6
## [59] yaml_2.1.18             zlibbioc_1.24.0
## [61] gplots_3.0.1            plyr_1.8.4
## [63] grid_3.4.2              blob_1.1.0
## [65] gdata_2.18.0            lattice_0.20-35
## [67] Biostrings_2.46.0       splines_3.4.2
## [69] annotate_1.56.1         locfit_1.5-9.1
## [71] pillar_1.2.1            reshape2_1.4.3
## [73] codetools_0.2-15        geneplotter_1.56.0
## [75] biomaRt_2.34.2          XML_3.98-1.10
## [77] glue_1.2.0              evaluate_0.10.1
## [79] latticeExtra_0.6-28     pcaMethods_1.70.0
## [81] tidyr_0.8.0             purrr_0.2.4
## [83] gtable_0.2.0            assertthat_0.2.0
## [85] ggplot2_2.2.1.9000      xtable_1.8-2
## [87] viridisLite_0.3.0       survival_2.41-3
## [89] tibble_1.4.2            GenomicAlignments_1.14.1
## [91] memoise_1.1.0           bindrcpp_0.2
## [93] cluster_2.0.6           LSD_4.0-0
## [95] BiocStyle_2.6.1
```