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

June 26, 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 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 methods to easily filter unexpressed genes and to analyse as well as to 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. *OUTRIDER* makes use of an autoencoder to control automatically for confounders within the data. A scheme of this approach is given in Figure **??**.

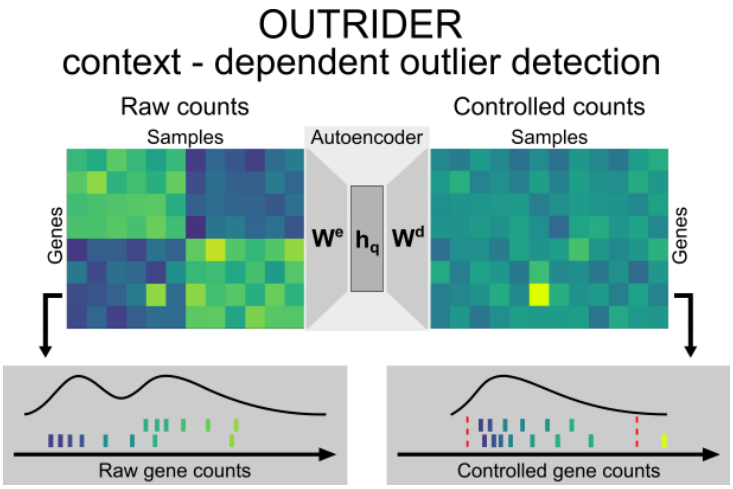


Figure 1: Context-dependent outlier detection
The algorithm identifies gene expression outliers whose read counts are significantly aberrant given the co-variations typically observed across genes in an RNA sequencing data set. This is illustrated by a read count (left panel, fifth column, second row 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.

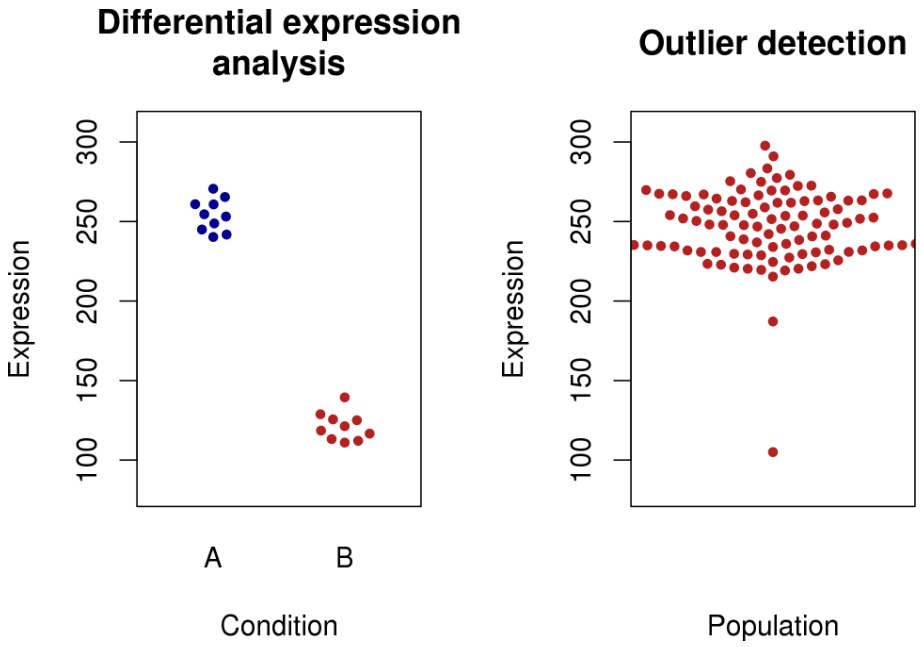


Figure 2: Scheme of workflow differences
Differences between differential gene expression analysis and outlier detection.

2 Prerequisites

To get started on the preprocessing step, we recommend to read the introductions of 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 (June 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. 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. 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
```

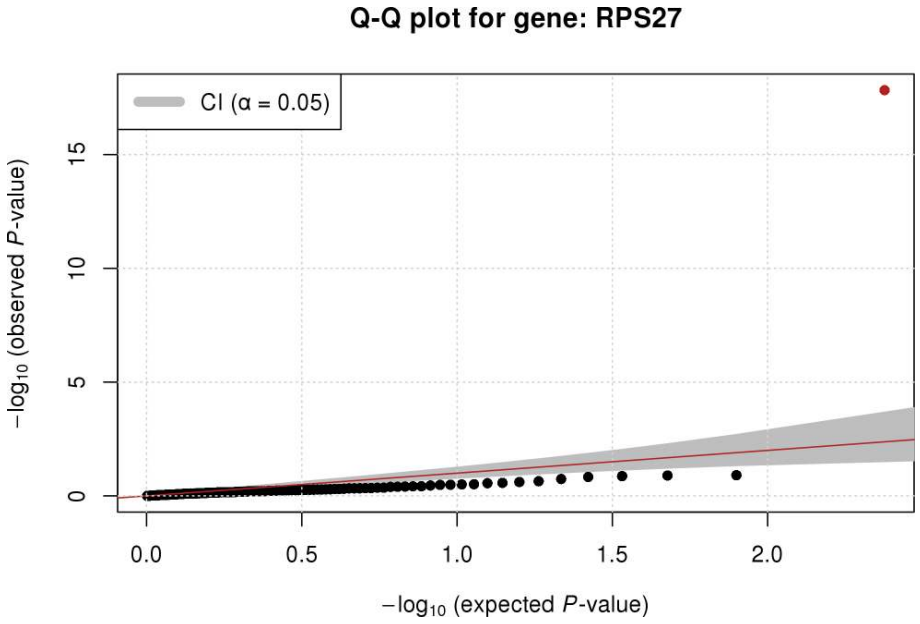
```
ctsFile <- system.file('extdata', 'KremerNBaderSmall.tsv',
        package='OUTRIDER')
ctsTable <- read.table(ctsFile, check.names=FALSE)
ods <- OutriderDataSet(countData=ctsTable)

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

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

```
## [1] "Initial PCA loss: 151.224099031229"
## Time difference of 4.330832 secs
## [1] "nb-PCA loss: 151.040171743756"
```

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

```
##     geneID sampleID       pValue      padjust zScore  l2fc rawcounts
## 1:   RPS27  MUC1372 1.484418e-18 8.416196e-15  -9.38 -2.31       949
## 2:   PARK7  MUC1372 1.066515e-15 3.023406e-12  -8.35 -0.97      1354
## 3:   UQCRH  MUC1372 5.059655e-15 9.562230e-12  -8.38 -1.55       490
## 4:   RPL11  MUC1372 7.522321e-15 1.066231e-11  -8.31 -1.53      2713
## 5:  MRPS21  MUC1372 2.986478e-13 3.386482e-10  -7.85 -1.30       300
## 6: LAMTOR2  MUC1372 4.315694e-13 4.078110e-10  -7.85 -1.27       185
##    normcounts aberrant mu   disp meanCorrected AberrantBySample
## 1:    2217.58     TRUE  1  49.83      10975.07               36
## 2:    2287.79     TRUE  1 191.98       4494.37               36
## 3:     901.78     TRUE  1  80.71       2643.27               36
## 4:    5417.63     TRUE  1  76.59      15599.16               36
## 5:     604.84     TRUE  1 104.71       1492.07               36
## 6:     349.56     TRUE  1 124.77        844.08               36
##    AberrantByGene padj_rank
## 1:              1         1
## 2:              1         2
## 3:              1         3
## 4:              1         4
## 5:              1         5
## 6:              1         6
```

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



4 An *OUTRIDER* analysis in detail

Additionally to the single wrapper function `OUTRIDER` to run the full pipeline, 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, this package contains also a small 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 at least one sample has one read (counts > 0) and that we observe one read 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.

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 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].

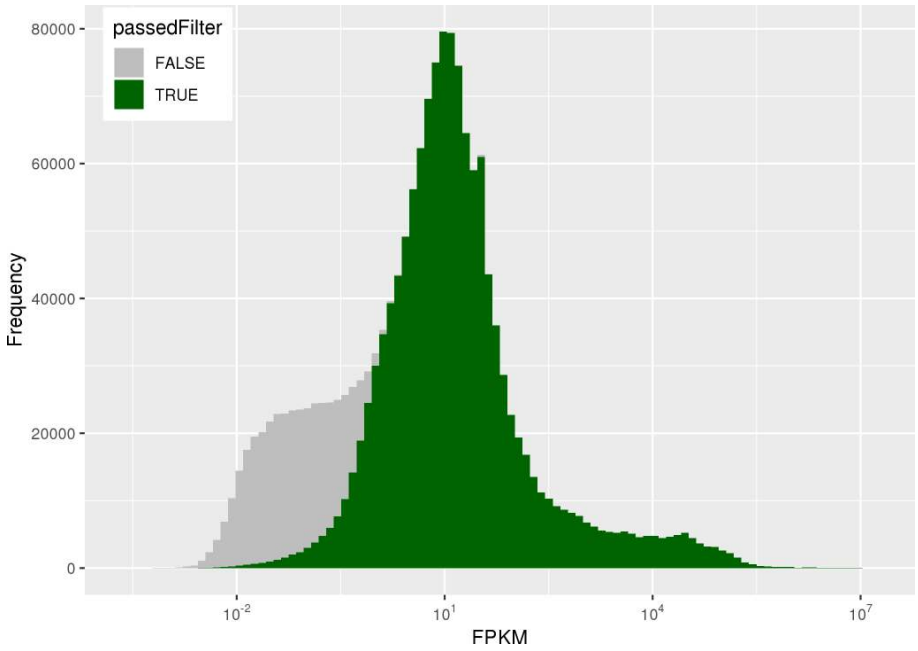
```
library(RMySQL)
library(AnnotationDbi)
txdbUrl <- paste0("https://i12g-gagneurweb.in.tum.de/public/",
        "paper/mitoMultiOmics/ucsc.knownGenes.db")
download.file(txdbUrl, "ucsc.knownGenes.db")
txdb <- loadDb("ucsc.knownGenes.db")
con <- dbConnect(MySQL(), host='genome-mysql.cse.ucsc.edu',
        dbname="hg19", user='genome')
map <- dbGetQuery(con, 'select kgId AS TXNAME, geneSymbol from kgXref')
```

```
# 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://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

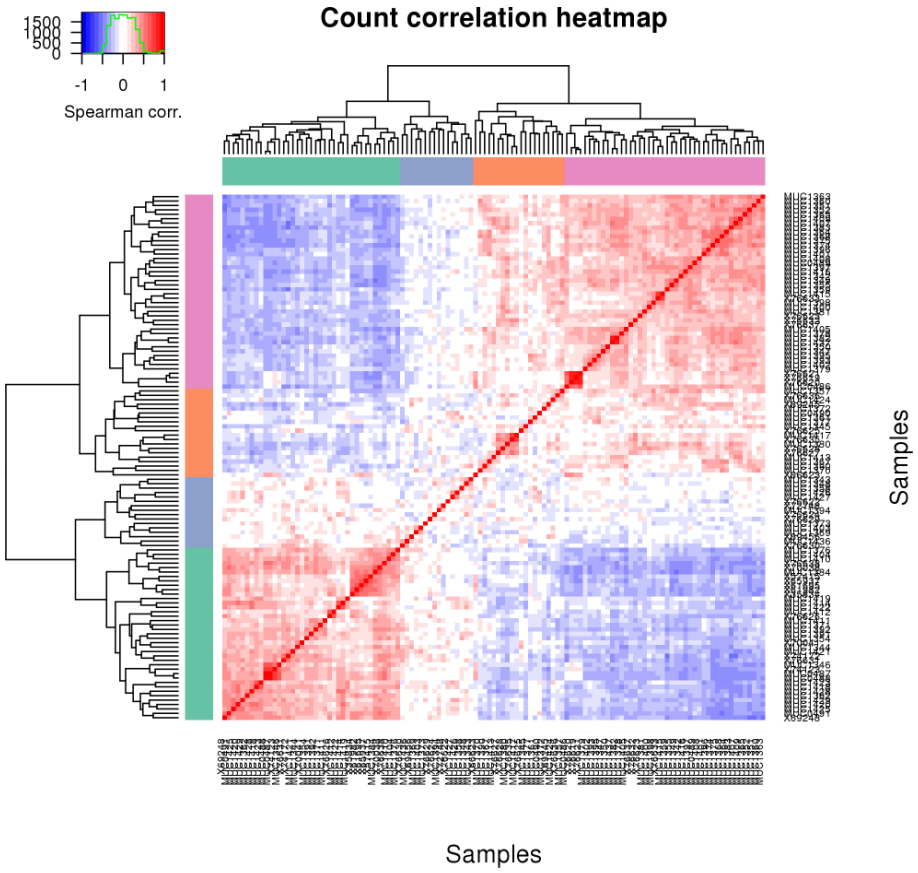



```
# do the actual subsetting based on the filtering labels
ods <- ods[mcols(ods)$passedFilter,]
```

4.2 Controlling for Confounders

A 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, which are often due to confounders: technical ones 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.

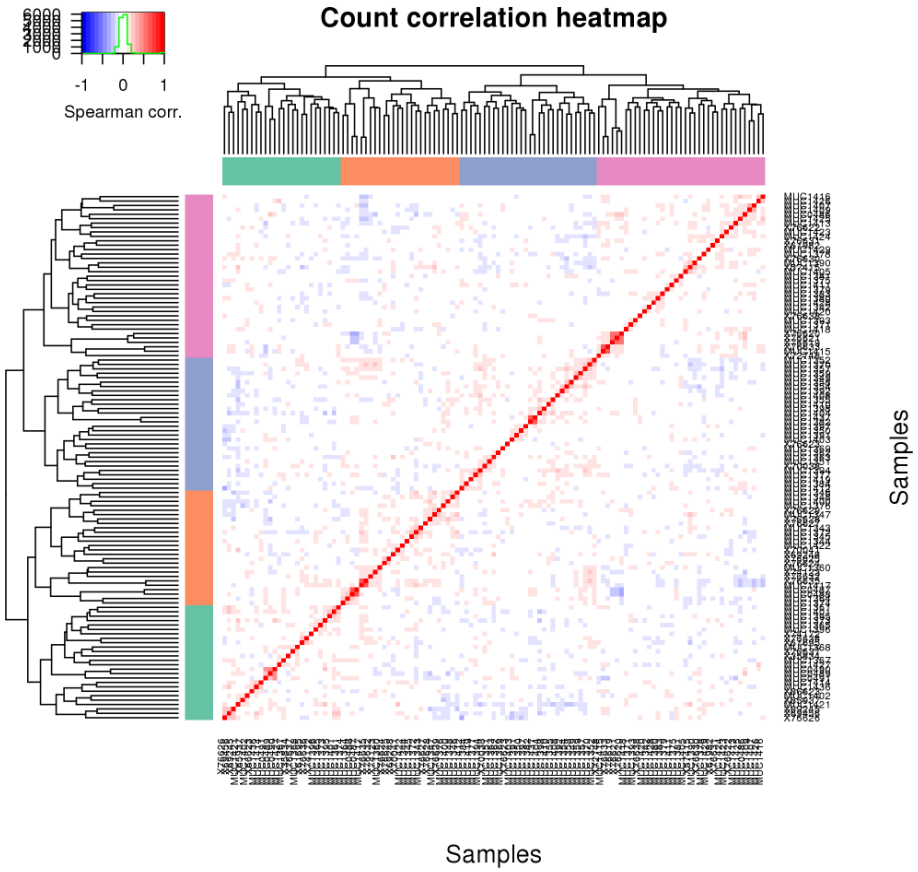
```
# Heatmap of the sample correlation
# it can also annotate the clusters resulting from the dendrogram
ods <- plotCountCorHeatmap(ods, normalized=FALSE, nCluster=4)
```



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*[1].

Additionally, the `autoCorrect` 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 determinde 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
ods <- estimateSizeFactors(ods)
ods <- autoCorrect(ods, q=13)

## [1] "Initial PCA loss: 180.52748840403"
## Time difference of 37.43726 secs
## [1] "nb-PCA loss: 180.414244422351"

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

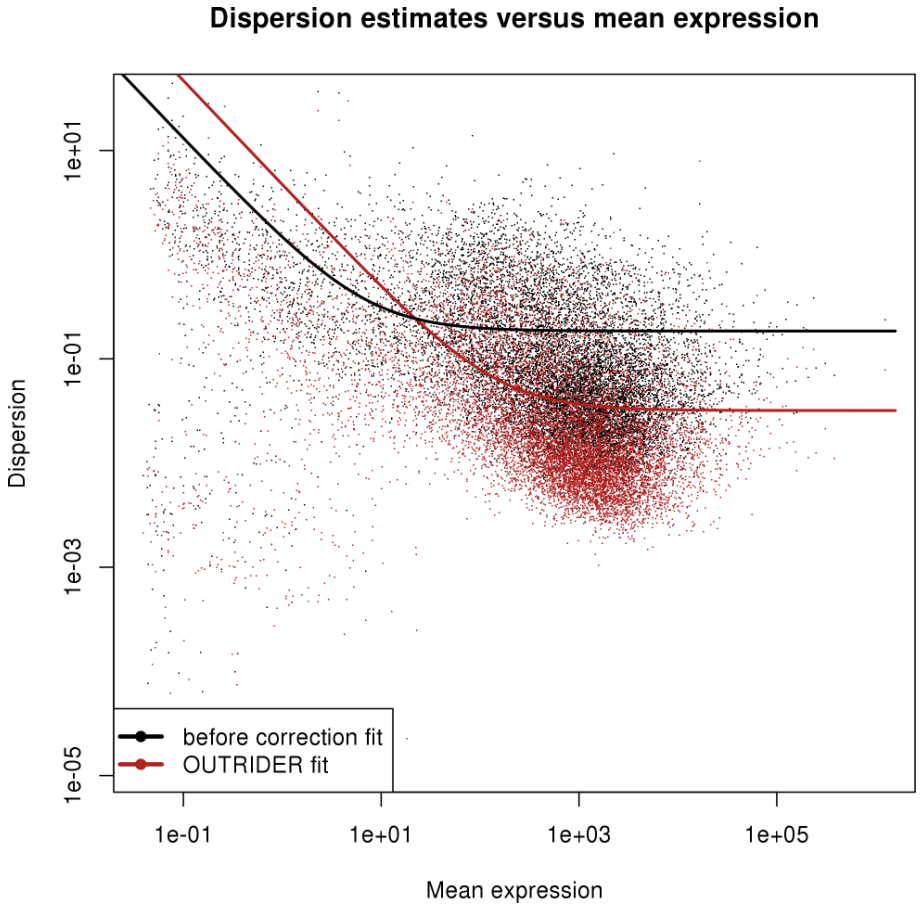


Alternatively, a `normalizationFactor` matrix can be provided. It must be computed beforehand using any method. Its purpose is to normalize for technical effects or control for additional expression patterns.

4.3 Fitting the negative binomial model

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

```
# fit negative binomial distribution to each feature
ods <- fit(ods)

# plot dispersion versus mean counts
plotDispEsts(ods)
```



4.4 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")
```

4.5 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:

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

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

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  l2fc rawcounts
## 1: NUDT12   X69456 5.470254e-21 6.287549e-16 -10.27 -8.52         0
## 2:  ATP5I  MUC1372 5.139671e-20 5.907575e-15  -9.74 -3.31        84
## 3:  STAG2   X76636 3.817586e-18 4.387961e-13  -9.23 -2.07       622
```

```
## 4: COX6B1  MUC1372 1.780025e-17 8.829806e-13  -9.04 -1.84       504
## 5:  RPS27  MUC1372 2.304616e-17 8.829806e-13  -9.19 -2.64       949
## 6:  COX7B  MUC1372 5.705830e-17 9.369032e-13  -8.95 -1.92       326
##    normcounts aberrant   mu  disp meanCorrected AberrantBySample
## 1:       0.00     TRUE 0.99 14.47        471.76                1
## 2:     138.07     TRUE 1.00 33.86       1380.04              868
## 3:     997.51     TRUE 1.00 59.13       4180.37                5
## 4:     679.90     TRUE 1.00 69.97       2441.26              868
## 5:    1761.55     TRUE 1.00 37.37      10965.55              868
## 6:     431.34     TRUE 1.00 64.84       1634.34              868
##    AberrantByGene padj_rank
## 1:              1       1.0
## 2:              1       1.0
## 3:              1       1.0
## 4:              1       2.5
## 5:              1       2.5
## 6:              1       5.5
```

```
dim(res)
```

```
## [1] 1054   15
```

```
# 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  l2fc rawcounts
## 1: NUDT12   X69456 5.470254e-21 6.287549e-16 -10.27 -8.52         0
## 2:  ATP5I  MUC1372 5.139671e-20 5.907575e-15  -9.74 -3.31        84
## 3:  STAG2   X76636 3.817586e-18 4.387961e-13  -9.23 -2.07       622
## 4: COX6B1  MUC1372 1.780025e-17 8.829806e-13  -9.04 -1.84       504
## 5:  RPS27  MUC1372 2.304616e-17 8.829806e-13  -9.19 -2.64       949
## 6:  COX7B  MUC1372 5.705830e-17 9.369032e-13  -8.95 -1.92       326
##    normcounts aberrant   mu  disp meanCorrected AberrantBySample
## 1:       0.00     TRUE 0.99 14.47        471.76                1
## 2:     138.07     TRUE 1.00 33.86       1380.04              868
## 3:     997.51     TRUE 1.00 59.13       4180.37                5
## 4:     679.90     TRUE 1.00 69.97       2441.26              868
## 5:    1761.55     TRUE 1.00 37.37      10965.55              868
## 6:     431.34     TRUE 1.00 64.84       1634.34              868
##    AberrantByGene padj_rank
## 1:              1       1.0
## 2:              1       1.0
## 3:              1       1.0
## 4:              1       2.5
## 5:              1       2.5
## 6:              1       5.5
```

```
dim(res)
```

```
## [1] 1220   15
```

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`).

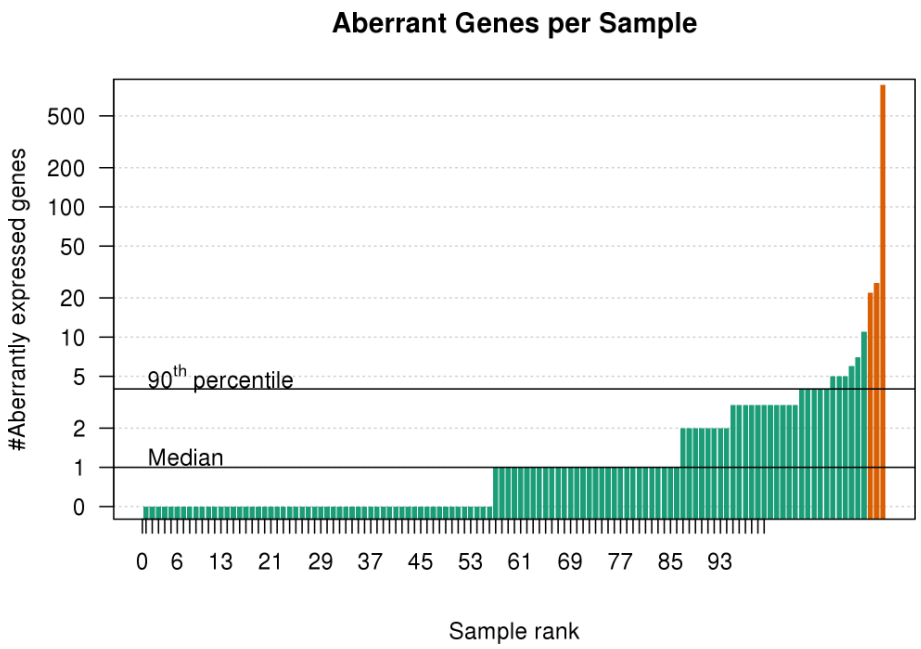
```
# number of aberrant genes per sample
tail(sort(aberrant(ods, by="sample")))
```

```
## MUC1347 MUC1362  X76630 MUC1421 MUC1403 MUC1372
##       6       7      11      22      26     868
```

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

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

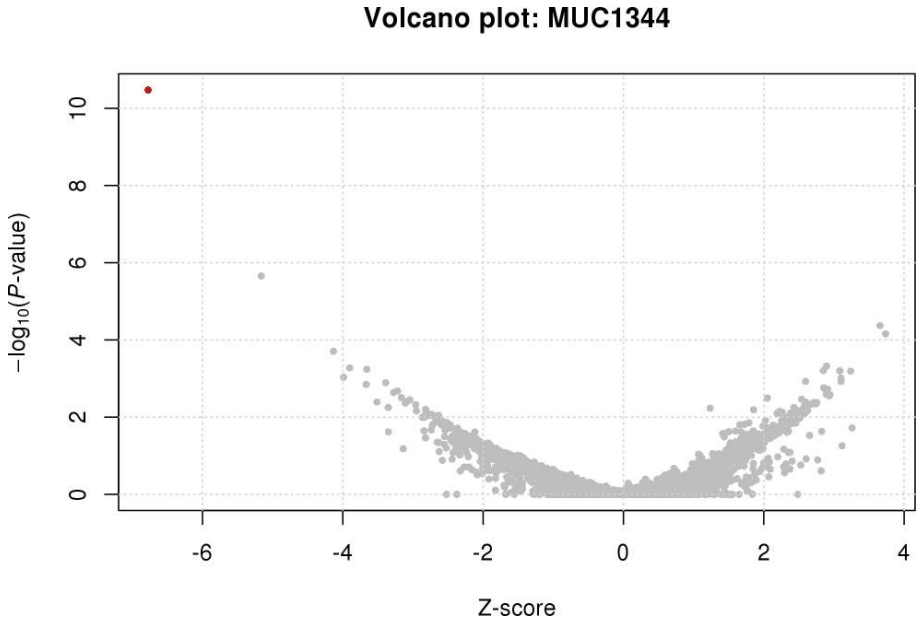
```
# plot the aberrant events per sample
plotAberrantPerSample(ods, padjCutoff=0.05)
```



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. For this vignette, we will always use the basic R functionality from *graphics* by setting the argument `basePlot` to `TRUE`.

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

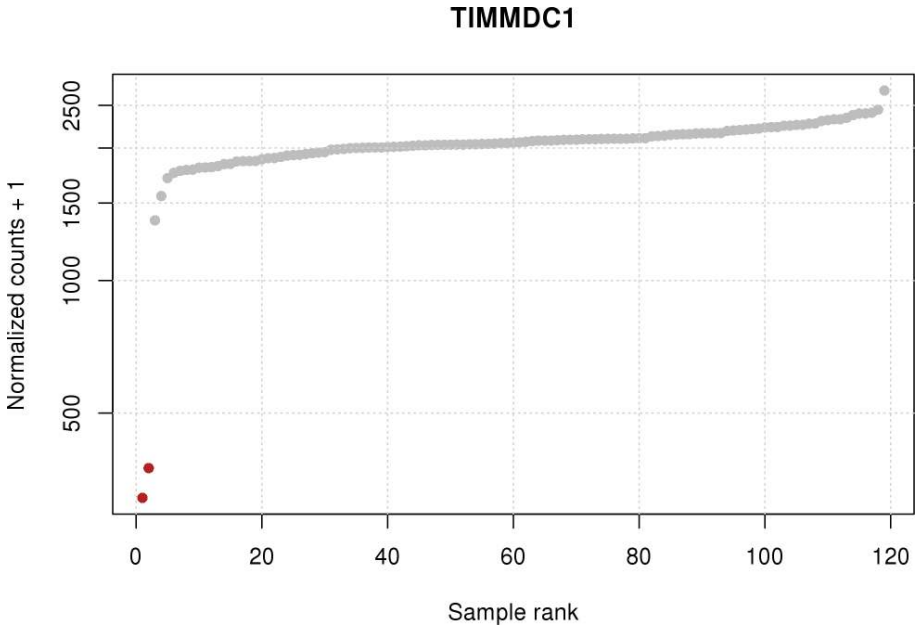


5.4 Gene level plots

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`.

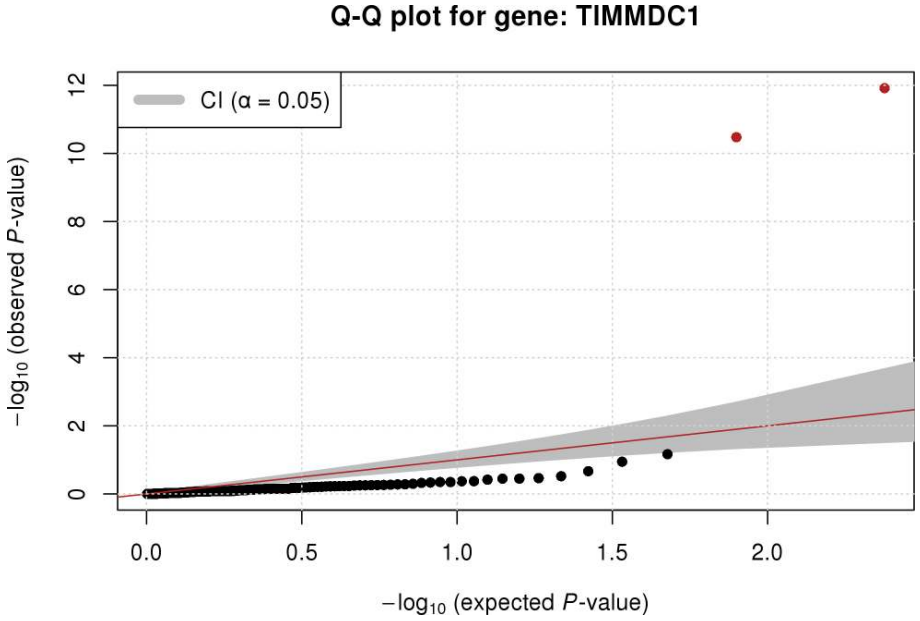
When using the *plotly* framework for plotting, all computed values are displayed for each data point. The user can access this information by hovering over each data point with the mouse.

```
# 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:

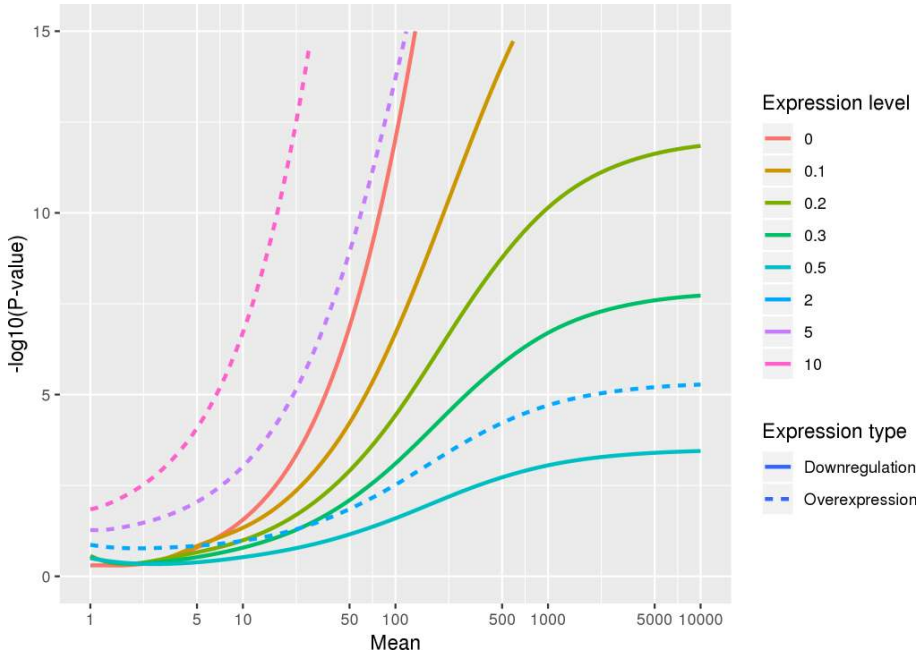
```
## QQ-plot for a given gene
plotQQ(ods, "TIMMDC1")
```



6 Details

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.

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 KoÅĽaÅŹíková, 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.

Session info

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

```
## R version 3.5.0 (2018-04-23)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Scientific Linux 7.5 (Nitrogen)
##
## Matrix products: default
## BLAS: /data/nasif12/modules_if12/SL7/i12g/R/3.5.0-Bioc3.7/lib64/R/lib/libRblas.so
## LAPACK: /data/nasif12/modules_if12/SL7/i12g/R/3.5.0-Bioc3.7/lib64/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
```

```
## [1] stats4    parallel  stats     graphics  grDevices datasets  utils
## [8] methods   base
##
## other attached packages:
##  [1] org.Hs.eg.db_3.6.0
##  [2] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [3] beeswarm_0.2.3
##  [4] OUTRIDER_0.99.7
##  [5] data.table_1.11.4
##  [6] SummarizedExperiment_1.10.1
##  [7] DelayedArray_0.6.1
##  [8] matrixStats_0.53.1
##  [9] GenomicFeatures_1.32.0
## [10] AnnotationDbi_1.42.1
## [11] Biobase_2.40.0
## [12] GenomicRanges_1.32.3
## [13] GenomeInfoDb_1.16.0
## [14] IRanges_2.14.10
## [15] S4Vectors_0.18.3
## [16] BiocGenerics_0.26.0
## [17] BiocParallel_1.14.1
## [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_3.5.0              backports_1.1.2
##  [9] R6_2.2.2                 rpart_4.1-13
## [11] KernSmooth_2.23-15       Hmisc_4.1-1
## [13] DBI_1.0.0                lazyeval_0.2.1
## [15] colorspace_1.3-2         nnet_7.3-12
## [17] tidyselect_0.2.4         gridExtra_2.3
## [19] prettyunits_1.0.2        DESeq2_1.20.0
## [21] bit_1.1-13               compiler_3.5.0
## [23] htmlTable_1.12           plotly_4.7.1
## [25] labeling_0.3             rtracklayer_1.40.2
## [27] caTools_1.17.1           scales_0.5.0.9000
## [29] checkmate_1.8.5          genefilter_1.62.0
## [31] stringr_1.3.1            digest_0.6.15
## [33] Rsamtools_1.32.0         foreign_0.8-70
## [35] rmarkdown_1.9            XVector_0.20.0
## [37] base64enc_0.1-3          pkgconfig_2.0.1
## [39] htmltools_0.3.6          highr_0.6
## [41] htmlwidgets_1.2          rlang_0.2.0.9001
```

```
## [43] rstudioapi_0.7          RSQLite_2.1.1
## [45] BBmisc_1.11             bindr_0.1.1
## [47] jsonlite_1.5            gtools_3.5.0
## [49] acepack_1.4.1           dplyr_0.7.5
## [51] RCurl_1.95-4.10         magrittr_1.5
## [53] GenomeInfoDbData_1.1.0  Formula_1.2-3
## [55] Matrix_1.2-14           Rcpp_0.12.17
## [57] munsell_0.5.0           reticulate_1.8
## [59] stringi_1.2.2           yaml_2.1.19
## [61] zlibbioc_1.26.0         gplots_3.0.1
## [63] plyr_1.8.4              grid_3.5.0
## [65] blob_1.1.1              gdata_2.18.0
## [67] lattice_0.20-35         Biostrings_2.48.0
## [69] splines_3.5.0           annotate_1.58.0
## [71] locfit_1.5-9.1          pillar_1.2.3
## [73] reshape2_1.4.3          geneplotter_1.58.0
## [75] biomaRt_2.36.1          XML_3.99-0
## [77] glue_1.2.0              evaluate_0.10.1
## [79] latticeExtra_0.6-28     pcaMethods_1.72.0
## [81] tidyr_0.8.1             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_2.42-3
## [89] tibble_1.4.2            GenomicAlignments_1.16.0
## [91] memoise_1.1.0           bindrcpp_0.2.2
## [93] cluster_2.0.7-1         BiocStyle_2.8.1
```