Contents

1 Introduction

An ever-growing variety of short read RNA sequencing methods is available to study the various aspects of transcript biosynthesis, processing and degradation (**???**). Some methods, such as mRNA sequencing (mRNA-seq), measure signals derived from processed mRNA and are thus ideally suitable for steady-state mature transcript level measurements and for the investigation of splice variants (**???**). Other techniques, such as Global Run-On sequencing (GRO-seq), nuclear RNA-seq (nucRNA-seq) and chromatin-associated RNA-seq (chrRNA-seq) provide information on primary transcription and give a more comprehensive picture of transcriptional activity, also for non-polyadenylated RNA (**???**; **???**; **???**; **???**). The differences in the RNA types being sequenced have an impact on the resulting sequencing profiles. mRNA-seq data is enriched with reads derived from exons, while GRO-, nucRNA- and chrRNA-seq demonstrate a substantial broader coverage of both exonic and intronic regions (**???**). The presence of intronic reads in GRO-seq type of data makes it possible to use it to computationally identify and quantify all *de novo* continuous regions of transcription distributed across the genome. This type of data, however, is more challenging to interpret and less common practice compared to mRNA-seq. One of the challenges for primary transcript detection concerns the simultaneous transcription of closely spaced genes, which needs to be properly divided into individually transcribed units. The R package *transcriptR* combines RNA-seq data with ChIP-seq data of histone modifications that mark active Transcription Start Sites (TSSs), such as, H3K4me3 or H3K9/14Ac to overcome this challenge. The advantage of this approach over the use of, for example, gene annotations is that this approach is data driven and therefore able to deal also with novel and case specific events. Furthermore, the integration of ChIP- and RNA-seq data allows the identification all known and novel active transcription start sites within a given sample.

transcriptR is an R package with a pipeline that is made up out of two main parts; an RNA-seq and a ChIP-seq part, of which the outputs are integrated to ultimately yield a comprehensive database of *de novo* identified primary transcripts and their abundance (Figure 1). In the first (RNA-seq) part, strand-specific GRO-seq, nucRNA-seq or chrRNA-seq short-reads in Binary Sequence Alignment Map (BAM) format are converted to coverage profiles. Background noise levels are estimated from regions with the low reads coverage using a Poisson-based approach. Genomic regions with read densities above background levels are considered to be expressed and small gaps in otherwise continuously transcribed regions are bridged when the gaps sizes are below a certain threshold which is extrapolated from the sequencing data and reference gene annotations. The second part, operates on ChIP-seq data and requires two input files: 1) a BAM file with the sequencing reads and 2) a peak file - output of a peak calling algorithm (for example MACS2) (**???**). As a first step, a classification model, based on the logistic regression, is used to predict and discriminate gene associated peaks from background peaks, using estimated characteristics of the peaks. Next, transcription initiation within a peak region is investigated by comparing RNA-seq read densities upstream and downstream of empirically determined transcription start sites. Putative transcription of both forward and reverse genomic strands is tested and the results are stored with each ChIP-seq peak.

At the end of the pipeline, both parts are combined and, where applicable, closely spaced transcripts are divided into individually transcribed units using the detected active transcription start sites. Additionally, the read count and FPKM value is calculated for each transcript in the dataset to facilitate further quantitative analysis.

The advantage of the two-part approach presented here is that the transcript detection and quantification can still be performed even in the absence of ChIP-seq data, bearing in mind that some adjacent transcripts may be detected as one transcribed unit.

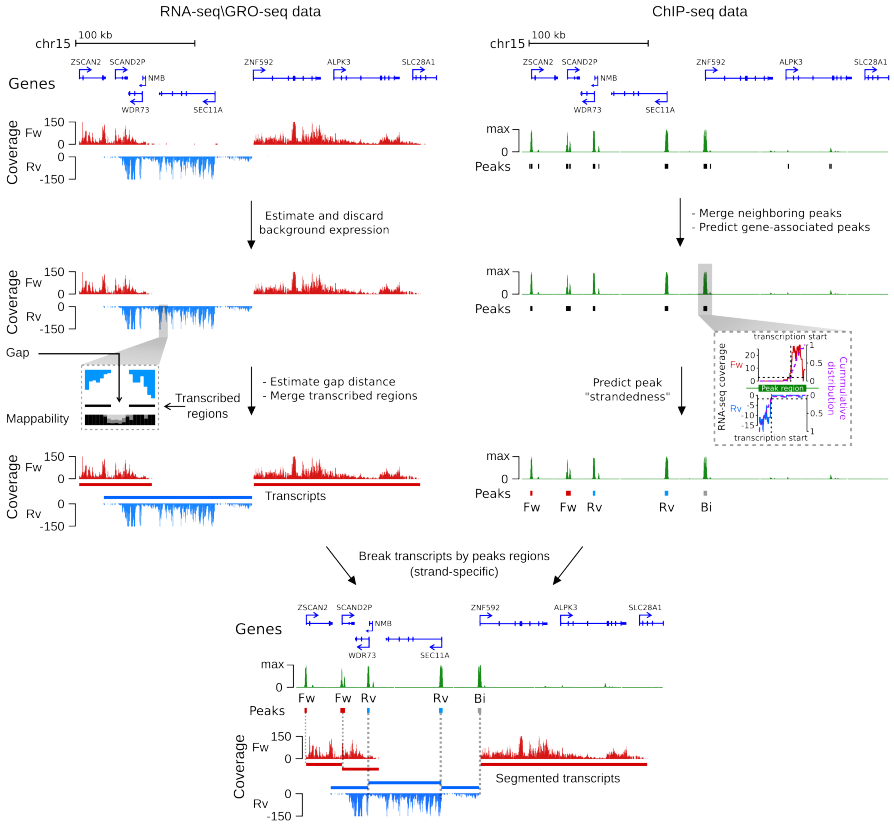

Figure 1: transcriptR workflow

In addition to the features mentioned above *transcriptR* provides functions for generating BigWig, BedGraph and BED files for the visualization of the coverage profiles, identified transcripts and peaks predictions in the UCSC genome browser.

2 Getting started

The *transcriptR* package is available in the `Bioconductor` and can be downloaded as follows:

```
> if (!requireNamespace("BiocManager", quietly=TRUE))
+     install.packages("BiocManager")
> BiocManager::install("transcriptR")
```

The following packages are required for the *transcriptR* workflow.

```
> library(GenomicFeatures)
> library(TxDb.Hsapiens.UCSC.hg19.knownGene)
> # Use UCSC genes as the reference annotations
```

```
> knownGene <- TxDb.Hsapiens.UCSC.hg19.knownGene
> # Extract genes information
> knownGene.genes <- genes(knownGene)
```

3 Transcripts detection

3.1 TranscriptionDataSet object construction

The workflow is initiated by creating a *TranscriptionDataSet* object, which is a container for holding processed sequencing data and the results of all downstream analyses. A function `constructTDS` initializes the construction of the *TranscriptionDataSet* object, by providing the paths to the input files and information relevant to the library preparation procedure. Optionally, the extracted reads can be limited to a specific genomic region, by using the `region` option. This would decrease the run time for the downstream analyses and might be useful for the testing purposes.

In this tutorial we supply an already constructed *TranscriptionDataSet*, containing nuclear RNA-seq data for human chromosome 15, profiled in prostate cancer LNCaP cells.

```
> # load TranscriptionDataSet object
> data(tds)
> # view TranscriptionDataSet object
> tds
## S4 object of class TranscriptionDataSet
## ========================================
## fragments: 1120345
## fragmentSize: 250
## coveragePlus: chr15
## coverageMinus: chr15
## coverageCutoff():
## gapDistance():
## transcripts(): 0
```

```
> # or initialize a new one (do not run the following code)
> # specify a region of interest (optional)
> region <- GRanges(seqnames = "chr15", ranges = IRanges(start = 63260000, end = 63300000))
> # object construction
> tds <- constructTDS(file = path.to.Bam,
+                     region = region,
+                     fragment.size = 250,
+                     unique = FALSE,
+                     paired.end = FALSE,
+                     swap.strand = TRUE)
```

Some of the supplied information can already be seen in the constructed *TranscriptionDataSet* object, whereas other slots are still empty and will be filled once the analysis is conducted.

3.2 Expression background estimation

Gene expression is a stochastic process, which often results in substantial expression noise. To obtain a putative set of transcribed regions, it is necessary to identify those regions that are expressed significantly above the background level. Using a Poisson-based approach for estimating the noise distribution, `estimateBackground` function returns a coverage cutoff value for a specific False Discovery Rate (FDR). The estimated value is stored in the `coverageCutoff` slot of the *TranscriptionDataSet* and will be used in the downstream analysis.

```
> estimateBackground(tds, fdr.cutoff = 0.01)
> # view estimated cutoff value
> tds
## S4 object of class TranscriptionDataSet
## ======================================
## fragments: 1120345
## fragmentSize: 250
## coveragePlus: chr15
## coverageMinus: chr15
## coverageCutoff(Params: fdr=0.01): 5.327
## gapDistance():
## transcripts(): 0
```

3.3 Coverage profile visualization

RNA-seq coverage profiles for both forward and reverse DNA strand can be visualized separately in the
UCSC genome browser using `exportCoverage`. This function can generate tracks in BigWig and bedGraph formats, which can be uploaded to the genome browser as custom tracks. There is an option to filter coverage profiles by a coverage cutoff value, either estimated for a specific FDR via `estimateBackground` or a user specified value. By default, the coverage cutoff value stored in the *TranscriptionDataSet* object is used. In order to make an informed decision about a proper FDR level, it is useful to explore the output at different FDR levels and determine the optimal cutoff value. Additionally, RPM (reads per million mapped reads) normalization is available.

```
> # look at the coverage profile of the regions expressed above the background level
> exportCoverage(object = tds, file = "plus.bw", type = "bigWig", strand = "+",
+                filter.by.coverage.cutoff = TRUE, rpm = FALSE)
>
> # or check the raw coverage (all expressed regions)
> exportCoverage(object = tds, file = "plus_raw.bw", type = "bigWig", strand = "+",
+                filter.by.coverage.cutoff = FALSE, rpm = FALSE)
```

3.4 Parametres tuning

The ultimate goal of *transcriptR* is to identify continuous regions of transcription. However, in some areas of the genome it is not possible to detect transcription, because of the presence of the low mappability regions and (high copy number) repeats. Sequencing reads can not be

uniquely mapped to these positions, leading to the formation of gaps in otherwise continuous coverage profiles and segmentation of transcribed regions into multiple smaller fragments. The gap distance describes the maximum allowed distance between adjacent fragments to be merged into one transcript. To choose the optimal value for the gap distance, the detected transcripts should largely be in agreement with available reference annotations. To accomplish this, the function is build on the methodology proposed by Hah et al. (**???**). In brief, this method uses two types of errors:

- 'dissected' error - the ratio of annotations that is segmented into two or more fragments.
- 'merged' error - the ratio of non-overlapping annotations that merged by mistake in the experimental data.

There is an interdependence between two types of errors. Increasing the gap distance decreases the 'dissected' error, by detecting fewer, but longer transcripts, while the 'merged' error will increase as more detected transcripts will span multiple annotations. The gap distance with the lowest sum of two error types is chosen as the optimal value.

The function `estimateGapDistance` uses increasing gap distances (based on the supplied vector) and calculates the associated error rates to determine the optimal gap distance.

```
> # create a range of gap distances to test
> # from 0 bp to 10000 bp with the step of 100 bp
> gd <- seq(from = 0, to = 10000, by = 100)
> estimateGapDistance(object = tds, annot = knownGene.genes, filter.annot = TRUE,
+                     fpkm.quantile = 0.25, gap.dist.range = gd)
## [INFO] Dissecting transcribed regions...Done!
## [INFO] Estimating gap distance minimizing sum of two errors...Done!
> # view the optimal gap distance minimazing the sum of two errors
> tds
## S4 object of class TranscriptionDataSet
## ======================================
## fragments: 1120345
## fragmentSize: 250
## coveragePlus: chr15
## coverageMinus: chr15
## coverageCutoff(Params: fdr=0.01): 5.327
## gapDistance(Params: coverage.cutoff=5.327): 6900 [error rate=0.238]
## transcripts(): 0
```

Additionally, all intermediate calculations can be accessed by `getTestedGapDistances` and the output can be presented in a graphical way by `plotErrorRate` function call. Here, the tested gap distances are plotted on the x-axis and corresponding error rates on the y-axis. Three curved lines depict the two error types calculated by `estimateGapDistance` and the sum of both errors. The vertical dashed line depicts the gap distance with the smallest sum of two errors.

```
> # get intermediate calculation
> gdTest <- getTestedGapDistances(tds)
> head(gdTest)
##   error.dissected error.merged sum.two.errors gap.distance
## 1       0.9457831 0.0004232356      0.9462064            0
## 2       0.8975904 0.0014871468      0.8990775          100
## 3       0.8795181 0.0028177833      0.8823359          200
```

```
## 4       0.8192771 0.0052770449      0.8245542          300
## 5       0.7590361 0.0083135392      0.7673497          400
## 6       0.7048193 0.0104166667      0.7152359          500
> # plot error rates
> plotErrorRate(tds, lwd = 2)
```

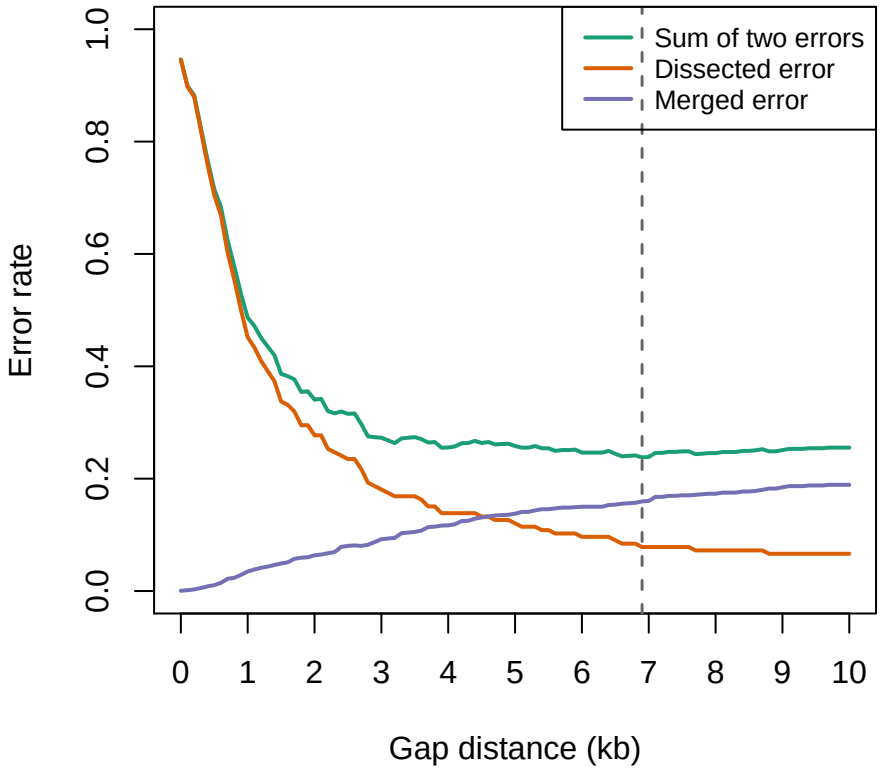

Figure 2: Gap distance error rate

3.5 Transcript calling

Transcript detection is performed by `detectTranscripts`. This function uses the two previously determined parameters to identify transcribed regions: `coverage.cutoff` and `gap.dist` as calculated by the `estimateBackground` and `estimateGapDistance`, respectively and stored in the *TranscriptionDataSet* object. Alternatively, the user may specify his/her own values to be passed to the function. By increasing the `gap.dist`, fewer transcripts of longer size will be identified, and an increase in the `coverage.cutoff` will result in fewer transcripts of shorter size.

```
> detectTranscripts(tds, estimate.params = TRUE)
## [INFO] Dissecting transcripts...Done!
## [INFO] Calculating fpkm and coverage...Done!
```

If desired, the identified transcripts can be associated with the available reference annotations based on the genomic overlap. To achieve this, the minimal proportion of the overlap between transcript and annotation is controlled by the `min.overlap` argument.

```
> annotateTranscripts(object = tds, annot = knownGene.genes, min.overlap = 0.5)
```

The detected transcripts can be retrieved by the `getTranscripts`. This function uses a number of arguments that control the resulting list of reported transcripts. In case, ChIP-seq data is not available, the following method will generates the final set of transcripts.

```
> trx <- getTranscripts(tds, min.length = 250, min.fpkm = 0.5)
> head(trx, 5)
## GRanges object with 5 ranges and 7 metadata columns:
##       seqnames            ranges strand |          id    length
##          <Rle>         <IRanges>  <Rle> | <character> <integer>
##   [1]    chr15 63337433-63339625      - |       trx_2      2193
##   [2]    chr15 63337613-63373090      + |       trx_3     35478
##   [3]    chr15 63414207-63464279      + |       trx_4     50073
##   [4]    chr15 63427619-63449482      - |       trx_5     21864
##   [5]    chr15 63482117-63614052      + |       trx_6    131936
##       bases.covered  coverage fragments      fpkm annotation.overlap
##           <integer> <numeric> <integer> <numeric>    <CharacterList>
##   [1]          1073     0.489        51    20.758                 ND
##   [2]         23919     0.674      5097   128.234               7168
##   [3]         19304     0.386      1299    23.155             114294
##   [4]         13461     0.616      2084    85.078              51065
##   [5]         93304     0.707      8108    54.853        51762,83464
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

3.6 Detected transcripts visualization

A convenient graphical way to explore the identified transcripts is to visualize them in the UCSC genome browser. The `transcriptsToBed` function returns a file in BED format, which can be directly uploaded to the genome browser. To improve the visual perception, transcripts are color-coded by DNA strand orientation.

```
> transcriptsToBed(object = trx, file = "transcripts.bed", strand.color = c("blue", "red"))
```

4 ChIP-seq peaks characterization and classification

When associated ChIP-seq data (e.g. H3K4me3 or H3K9ac) is available, this information can be used to identify the transcript starts. This is particularly useful in situations where genes are densely packed in the genome and the identification of individual transcripts from RNA-seq data is challenging. The inclusion of ChIP-seq peak information has to be performed carefully as not all identified peaks show evidence of active transcription in RNA-seq data. In order to discriminate between peaks with active transcription and background peaks, we use a two step approach. First, we collect the characteristics of ChIP-seq peaks that overlap transcription start sites of annotated genes. These characteristics are used to identify all putatively gene-associated ChIP-seq peaks. A schematic overview of the procedure is shown in Figure 3.

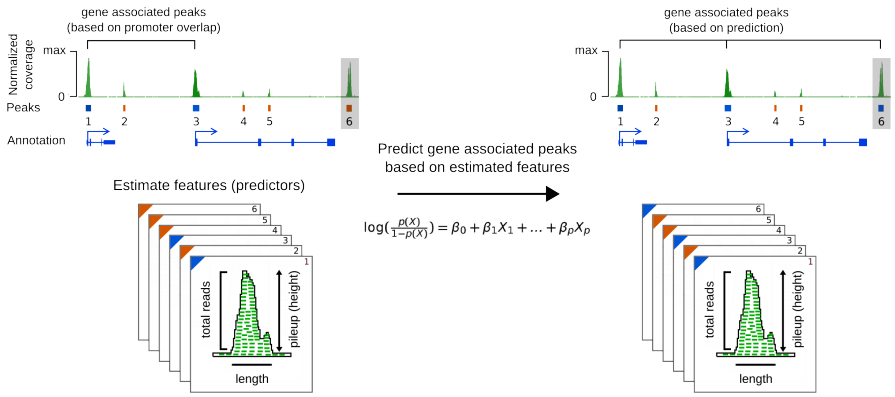

Figure 3: **Classification of the peaks on gene associated and background**

Secondly, we incorporate RNA-seq data to find direct evidence of active transcription from every putatively gene-associated peak. In order to do this, we determine the "strandedness" of the ChIP-seq peaks, using strand specific RNA-seq data. The following assumptions are made in order to retrieve the peak "strandedness":

- The putatively gene-associated ChIP-seq peaks are commonly associated with transcription initiation.
- This transcription initiation occurs within the ChIP peak region.
- When a ChIP peak is associated with a transcription initiation event, we expect to see a strand-specific increase in RNA-seq fragment count downstream the transcription initiation site (Figure 4).

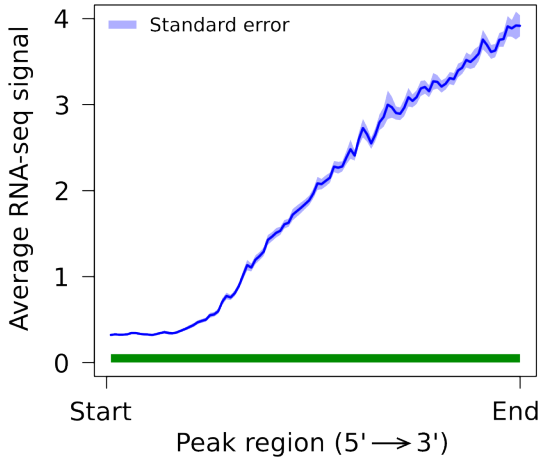

Figure 4: **Average RNA-seq signal across ChIP peak region**

Each peak in the data set is tested for association with transcription initiation on both strands of DNA. A detailed overview of the procedure is depicted in the Figure 5. Steps 1-5 are performed for both forward and reverse DNA strand separately and step 6 combines the data from both strands. If the peak is identified as associated with the transcription on both strands, than it is considered to be a bidirectional.

ChIP peak "strandedness" prediction steps:

1. Identify a location within the ChIP-seq peak near the transcription start site. This is accomplished by calculating the cumulative distribution of RNA-seq fragments within a peak region. The position is determined where 90% of RNA-seq fragments are located downstream. This approach performs well on both gene-poor and gene-dense regions where transcripts may overlap.
2. Two equally sized regions are defined (q1 and q2), flanking the position identified in (1) on both sides. RNA-seq fragments are counted in each region.
3. ChIP peaks with an RNA-seq fragment coverage below an estimated threshold are discarded from the analysis.
4. The probability is calculated for RNA-seq fragments to be sampled from either q1 or q2. Based on the assumptions we stated above, a ChIP peak that is associated with transcription initiation should have more reads in q2 (downstream of the transcription start position) compared to q1, and subsequently, the probability of a fragment being sampled from q2 would be higher.
5. ChIP-seq peaks are divided into gene associated and background based on the prediction.
6. Iteratively, the optimal P(q2) threshold is identified, which balances out the False Discovery Rate (FDR) and False Negative Rate (FNR). Peaks with the P(q2) exceeding the estimated threshold are considered to be associated with the transcription initiation event.

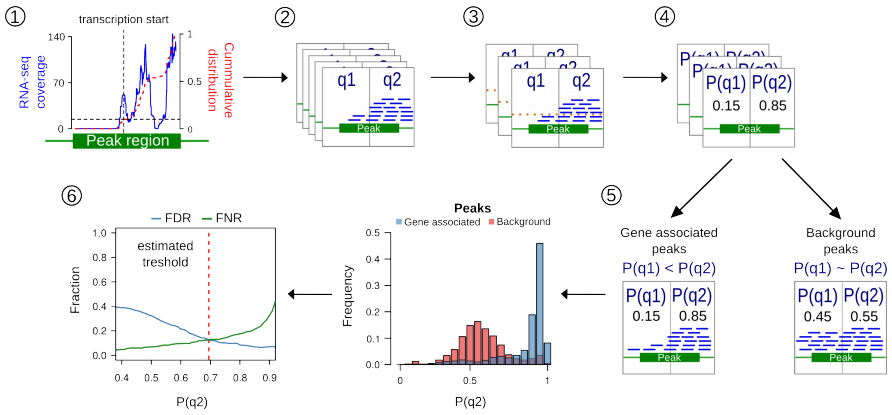

Figure 5: ChIP peak "strandedness" prediction

4.1 ChipDataSet object construction

We start ChIP-seq analysis by creating a *ChipDataSet* object, which is a container for storing ChIP-seq peaks information and all the results that will be generated by applying specific functions. The function `constructCDS` initializes the *ChipDataSet* object, by providing the paths to the input files and information relevant to the ChIP-seq library preparation procedure. During the object construction the following steps are executed:

- The peak information is converted into the object of *GRanges* class.
- The genomic distribution of the peaks is evaluated (exonic, intronic, intergenic, TSSs).

- Each peak in the data set is functionally characterized (peak length, total number of reads, pileup, etc.). The estimated features are used to predict which of the peaks are gene associated in the analysis downstream.

As many peak-calling algorithms tend to divide broader peaks into the several narrower closely spaced peaks, it is advised to merge these end-to-end peaks to decrease the number of false positives and prevent unnecessary truncation of transcripts in the downstream analysis.

In this tutorial we supply an already constructed *ChipDataSet*, containing H3K4me3 active histone mark ChIP-seq peaks from the chromosome 15, profiled in the prostate cancer LNCaP cells.

```
> # load ChipDataSet object
> data(cds)
> cds
## S4 object of class ChipDataSet
## ====================================
## Peaks: 383
## Chromosome distribution: chr15
## Features:  tssOverlap length fragments density pileup
## Gene associated peaks (predicted):
## Gene associated peaks by strand:
```

```
> # or initialize a new one (do not run the following code)
> # specify the region of interest (optional)
> region <- GRanges(seqnames = "chr15", ranges = IRanges(start = 63260000, end = 63300000))
> # object construction
> cds <- constructCDS(peaks = path.to.peaks,       # path to a peak file
+                     reads = path.to.reads,       # path to a Bam file with reads
+                     region = region,
+                     TxDb = knownGene,            # annotation database to evaluate
+                                                  # genomic distribution of the peaks
+                     tssOf = "transcript",        # genomic feature to extract TSS region
+                     tss.region = c(-2000, 2000), # size of the TSS region,
+                                                  # from -2kb to +2 kb
+                     reduce.peaks = TRUE,         # merge neighboring peaks
+                     gapwidth = 500,              # min. gap distance between peaks
+                     unique = TRUE,
+                     swap.strand = FALSE)
```

The peaks stored in *ChipDataSet* can be retrieved by a simple `getPeaks` function call.

```
> peaks <- getPeaks(cds)
> head(peaks, 3)
## GRanges object with 3 ranges and 7 metadata columns:
##       seqnames            ranges strand |                  merged.peaks
##          <Rle>         <IRanges>  <Rle> |                <CharacterList>
##   [1]    chr15 63339894-63343845      * |                MACS_peak_12700
##   [2]    chr15 63345606-63345862      * |                MACS_peak_12701
##   [3]    chr15 63412975-63415230      * | MACS_peak_12702,MACS_peak_12703
##                id tssOverlap    length fragments   density    pileup
##       <character>   <factor> <integer> <integer> <numeric> <integer>
##   [1]      peak_1        yes      3952      1138     0.288        33
```

```
##   [2]      peak_2         no       257        25     0.097         7
##   [3]      peak_3        yes      2256       770     0.341        35
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

A simple quality check of the supplied ChIP-seq peaks can be performed by investigating their genomic distribution. Ideally, these peaks should demonstrate substantial enrichment at TSS regions. Enrichment of the peaks at a given genomic feature (e.g. TSS) is defined as the ratio between the observed and expected number of peaks. The expected number of peaks is calculated from the proportion of the genome covered by the given genomic feature.

```
> getGenomicAnnot(cds)
##             region observed expected     ratio
## 1 TSS [-2000:2000]      176       19 9.2631579
## 2            Exons        6        8 0.7500000
## 3          Introns      123      144 0.8541667
## 4       Intergenic       78      212 0.3679245
```

Additionally, the genomic distribution of the peaks can be visualized in two ways, either by observing the total number of peaks overlapping given genomic feature (Figure 6) or by looking at the enrichment levels (Figure 7).

```
> plotGenomicAnnot(object = cds, plot.type = "distr")
```

```
> plotGenomicAnnot(object = cds, plot.type = "enrich")
```

4.2 Prediction of the gene associated peaks

In order to discriminate between functional or gene associated peaks and non-functional or background peaks, each peak in the data set is characterized by several features. Among them:

- `length` - the length of a peak (in base pairs).
- `fragments` - total number of fragments overlapping a peak region.
- `density` - number of fragments per base pair of the peak length.
- `pileup` - highest fragment pileup in each peak region.
- `tssOverlap` - overlap (binary, yes/no) of the peak with the annotated TSS region.

Moreover, the user may wish to supply her/his own list of features with the `addFeature`. See the manual page for an example of the use of `addFeature`.

Prior to fitting the logistic model, the relations between predictors and response variable (`tssOverlap`) can be explored with `plotFeatures` (Figure 8). Based on the plots, poor predictors can be excluded from the analysis to improve the model fit. In general, the `pileup` (peak height) is a good predictor on its own, as it can segregate gene associated and background peaks very well.

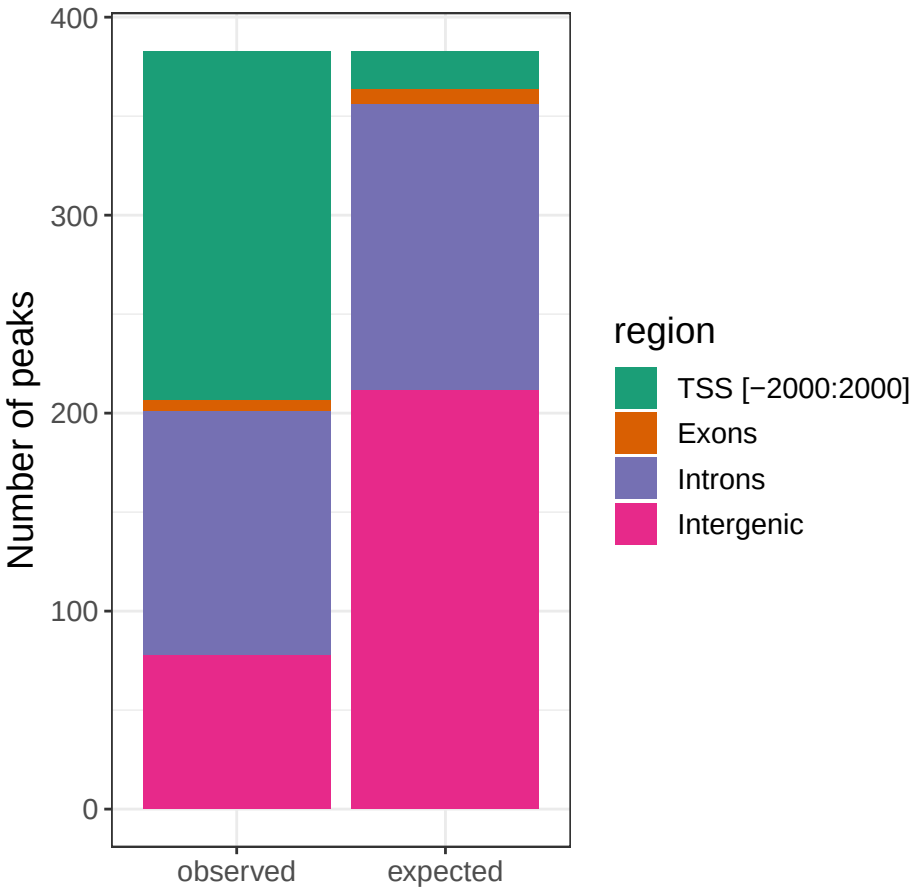

Figure 6: Genomic distribution of the peaks

```
> plotFeatures(cds, plot.type = "box")
## Warning: `panel.margin` is deprecated. Please use `panel.spacing` property
## instead
```

The logistic regression model fit is accomplished by `predictTssOverlap`. In order to improve the accuracy of the model the data is internally partitioned into a training and testing data sets. The percent of the data that will be allocated to the training set should be specified by parameter `p`. A repeated 10-Fold Cross-Validation is used to calculate performance measures on the training data set and to prevent over-fitting. It is possible to specify a subset of features (predictors) to be used in the model fit via the `feature` argument. By default, all the features will be used.

```
> predictTssOverlap(object = cds, feature = "pileup", p = 0.75)
## [INFO] Partitioning data...Done!
## [INFO] Training model:
##        Accuracy - 0.875
##        Area under the curve (AUC) - 0.9259
## [INFO] Testing model:
##        Accuracy - 0.8842
```

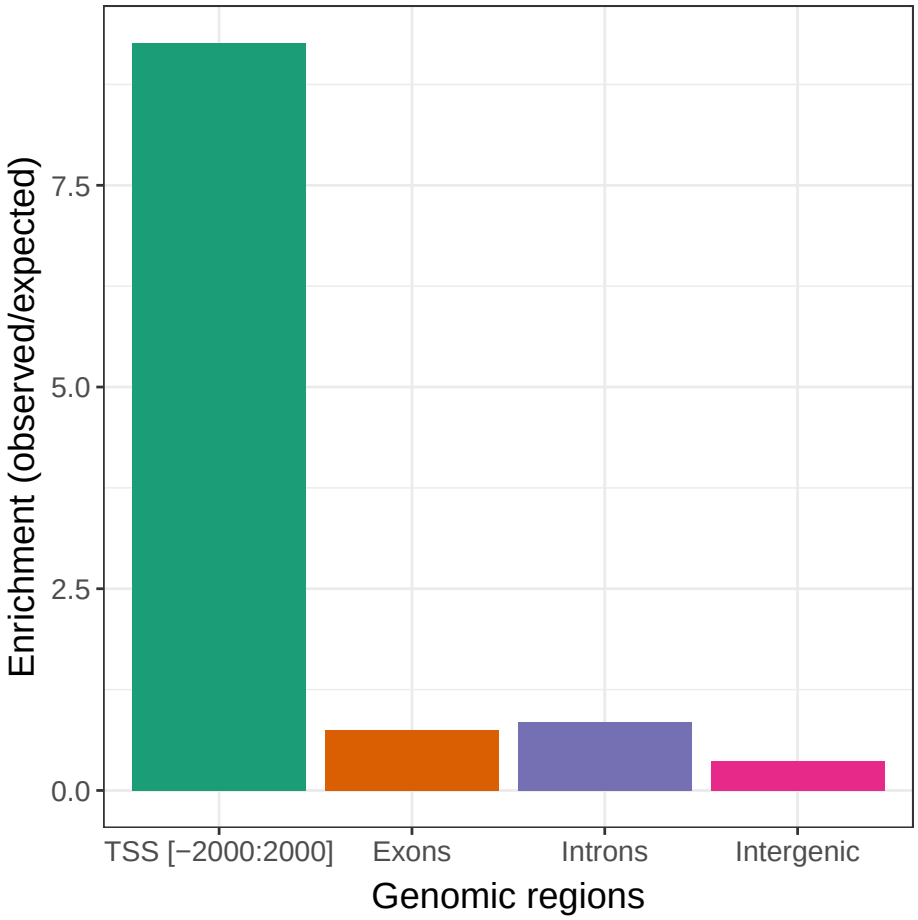

Figure 7: Enrichment of peaks at distinct genomic features

```
##        Area under the curve (AUC) - 0.9205
> cds
## S4 object of class ChipDataSet
## ======================================
## Peaks: 383
## Chromosome distribution: chr15
## Features:  tssOverlap length fragments density pileup
## Gene associated peaks (predicted):  155
## Gene associated peaks by strand:
```

The result of the logistic regression model fit, can be viewed by `getPeaks`. Two new columns have been added to the output: 1) `predicted.tssOverlap.prob` - estimated probability of a peak being gene associated and 2) `predicted.tssOverlap` - predicted class (binary, yes/no).

```
> peaks <- getPeaks(cds)
> head(peaks, 3)
## GRanges object with 3 ranges and 9 metadata columns:
##       seqnames            ranges strand |                 merged.peaks
##          <Rle>         <IRanges>  <Rle> |               <CharacterList>
```

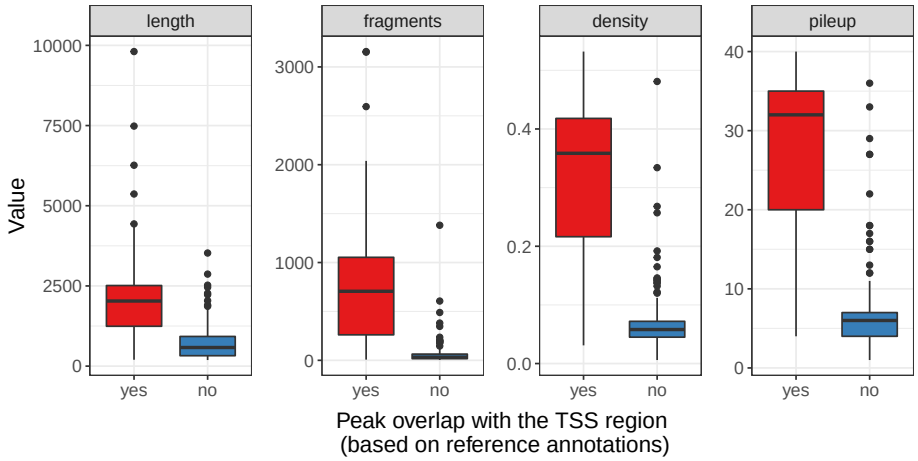

Figure 8: Estimated features

```
##   [1]    chr15 63339894-63343845      * |                MACS_peak_12700
##   [2]    chr15 63345606-63345862      * |                MACS_peak_12701
##   [3]    chr15 63412975-63415230      * | MACS_peak_12702,MACS_peak_12703
##                id tssOverlap    length fragments   density    pileup
##       <character>   <factor> <integer> <integer> <numeric> <integer>
##   [1]      peak_1        yes      3952      1138     0.288        33
##   [2]      peak_2         no       257        25     0.097         7
##   [3]      peak_3        yes      2256       770     0.341        35
##       predicted.tssOverlap.prob predicted.tssOverlap
##                       <numeric>             <factor>
##   [1]         0.982735651274388                  yes
##   [2]         0.169221670148759                   no
##   [3]         0.988737972805949                  yes
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The detailed overview of the classification model fit to the data and the significance of each predictor can be accessed by `getConfusionMatrix` and `getPredictorSignificance` respectively. Features with a high p-value can be excluded and the analysis should be rerun with the significant predictors only.

```
> getConfusionMatrix(cds)
## Confusion Matrix and Statistics
##
##           Reference
## Prediction yes  no
##        yes 142  13
##        no   34 194
##
##                Accuracy : 0.8773
##                  95% CI : (0.8402, 0.9084)
##     No Information Rate : 0.5405
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                  Kappa : 0.7507
##
##  Mcnemar's Test P-Value : 0.003531
##
##             Sensitivity : 0.8068
##             Specificity : 0.9372
##          Pos Pred Value : 0.9161
##          Neg Pred Value : 0.8509
##              Prevalence : 0.4595
##          Detection Rate : 0.3708
##    Detection Prevalence : 0.4047
##       Balanced Accuracy : 0.8720
##
##        'Positive' Class : yes
##
> getPredictorSignificance(cds)
## [1] 3.90338e-16
```

Additionally, the performance of the model can be visualized by a Receiver operating characteristic (ROC) plot (Figure 9). The curve is created by plotting the true positive rate (sensitivity) against the false positive rate (1 - specificity) at various threshold settings. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test. The area under the curve (AUC) is a measure of accuracy.

```
> plotROC(object = cds, col = "red3", grid = TRUE, auc.polygon = TRUE)
```

4.3 Prediction of the peak strandedness

The Prediction of the peak "strandedness" is accomplished by `predictStrand`. This function requires information stored in both *ChipDataSet* and *TranscriptionDataSet* objects.

```
> predictStrand(cdsObj = cds, tdsObj = tds, quant.cutoff = 0.1, win.size = 2000)
## [INFO] Creating coverage profiles for the starts of the fragments...Done!
## [IFNO] Estimating transcription start position...Done!
## [INFO] Calculating probabilities...Done!
## [INFO] Estimating probability cutoff value...Done!
## [INFO] Finalizing output...Done!
> cds
## S4 object of class ChipDataSet
## ====================================
## Peaks: 383
## Chromosome distribution: chr15
## Features:  tssOverlap length fragments density pileup
## Gene associated peaks (predicted):  155
## Gene associated peaks by strand:  48(+) 36(-) 38(bi) 33(.)
> peaks <- getPeaks(cds)
> head(peaks[ peaks$predicted.tssOverlap == "yes" ], 3)
## GRanges object with 3 ranges and 10 metadata columns:
##       seqnames            ranges strand |                  merged.peaks
##          <Rle>         <IRanges>  <Rle> |                <CharacterList>
```

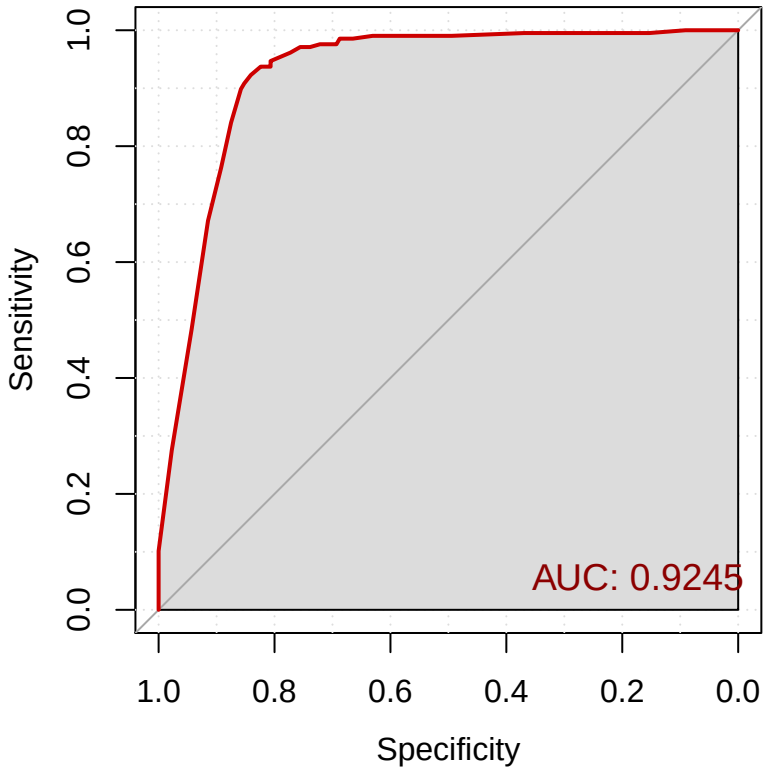

Figure 9: Performance of the classification model (gene associated peaks prediction)

```
##   [1]    chr15 63339894-63343845      * |                MACS_peak_12700
##   [2]    chr15 63412975-63415230      * | MACS_peak_12702,MACS_peak_12703
##   [3]    chr15 63448191-63450653      * |                MACS_peak_12704
##               id tssOverlap    length fragments   density    pileup
##      <character>   <factor> <integer> <integer> <numeric> <integer>
##   [1]      peak_1        yes      3952      1138     0.288        33
##   [2]      peak_3        yes      2256       770     0.341        35
##   [3]      peak_4        yes      2463      1311     0.532        37
##      predicted.tssOverlap.prob predicted.tssOverlap predicted.strand
##                        <numeric>             <factor>         <factor>
##   [1]         0.982735651274388                  yes                +
##   [2]         0.988737972805949                  yes                +
##   [3]         0.992669025676404                  yes               bi
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

The `predicted.strand` can take one of the four possible variants: 1) "+" plus strand, 2) "-" minus strand, 3) "bi" bidirectional and 4) "." not associated with a transcription initiation event. All internal calculations performed by `predictStrand` can be viewed by `getQuadProb`. It returns a data frame, where each row corresponds to a peak and each column keeps one of the intermediate calculations:

- `max.cov` - maximum coverage of the RNA-seq fragments inside the peak region.

- `pass.cov.treshold` - whether the `max.cov` exceeds the `coverage.cutoff`, either user defined or estimated from RNA-seq data by `estimateBackground` function call and stored in *TranscriptionDataSet* object.
- `q1q2.sepline.coord` - genomic coordinate corresponding to the transcription start position inside the peak region and which is used to separate q1 and q2.
- `q1.coord` - genomic coordinates of q1.
- `q2.coord` - genomic coordinates of q2.
- `q1.count` - total number of fragments in q1.
- `q2.count` - total number of fragments in q2.
- `q1.prob` - probability of a fragment being sampled from the q1.
- `q2.prob` - probability of a fragment being sampled from the q2 .

```
> df <- getQuadProb(cds, strand = "+")
> head(df, 3)
##   max.cov pass.cov.treshold q1q2.sepline.coord               q1.coord
## 1      52               yes           63340883 chr15:63338882-63340882
## 2      49               yes           63345659 chr15:63343658-63345658
## 3      15               yes           63414196 chr15:63412195-63414195
##                   q2.coord q1.count q2.count    q1.prob   q2.prob
## 1 chr15:63340883-63342883       46      197 0.18930041 0.8106996
## 2 chr15:63345659-63347659      132      148 0.47142857 0.5285714
## 3 chr15:63414196-63416196        3       38 0.07317073 0.9268293
```

The predicted P(q2) threshold, used to select peaks with a putative transcription initiation event, is accessed by `getProbTreshold`.

```
>  getProbTreshold(cds)
## [1] 0.64
```

4.4 Peaks visualization

A convenient way to explore output of the predictions made on the ChIP peaks is to visualize them in the UCSC genome browser. The `peaksToBed` function returns a file in BED format, which can be uploaded directly to the genome browser. To improve the visual perception, peaks are color-coded by the predicted strand.

```
> peaksToBed(object = cds, file = "peaks.bed", gene.associated.peaks = TRUE)
```

5 Transcript boundaries demarcation

As a last step, both parts of the workflow are combined and, where applicable, closely spaced transcripts are divided into individually transcribed units using the detected active transcription start sites. There is a single function `breakTranscriptsByPeaks`, which will generate the final set of transcripts.

```
> # set `estimate.params = TRUE` to re-calculate FPKM and coverage density
> breakTranscriptsByPeaks(tdsObj = tds, cdsObj = cds, estimate.params = TRUE)
## [IFNO] Breaking transcripts by peaks...Done!
## [INFO] Calculating fpkm and coverage ...Done!
> # re-annotate identified transcripts
> annotateTranscripts(object = tds, annot = knownGene.genes, min.overlap = 0.5)
> # retrieve the final set of transcripts
> trx.final <- getTranscripts(tds)
```

```
> # visualize the final set of transcripts in a UCSC genome browser
> transcriptsToBed(object = trx.final, file = "transcripts_final.bed")
```

We can explore which transcripts are broken by peaks, by simply intersecting two sets of transcript outputs (before breaking transcripts by peaks, and after).

```
> hits <- findOverlaps(query = trx, subject = trx.final)
> trx.broken <- trx[unique(queryHits(hits)[duplicated(queryHits(hits))])]
> head(trx.broken, 5)
## GRanges object with 5 ranges and 7 metadata columns:
##       seqnames            ranges strand |          id    length
##          <Rle>         <IRanges>  <Rle> | <character> <integer>
##   [1]    chr15 63337613-63373090      + |       trx_3     35478
##   [2]    chr15 63414207-63464279      + |       trx_4     50073
##   [3]    chr15 63482117-63614052      + |       trx_6    131936
##   [4]    chr15 64386513-64463030      + |      trx_18     76518
##   [5]    chr15 64442210-64679054      - |      trx_19    236845
##       bases.covered  coverage fragments      fpkm annotation.overlap
##           <integer> <numeric> <integer> <numeric>    <CharacterList>
##   [1]         23919     0.674      5097   128.234              7168
##   [2]         19304     0.386      1299    23.155            114294
##   [3]         93304     0.707      8108    54.853       51762,83464
##   [4]         55985     0.732      9277   108.216         6642,79856
##   [5]        174387     0.736     21005     79.16    53944,5479,9768
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

6 Session Information

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

```
## R version 3.6.0 (2019-04-26)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: OS X El Capitan 10.11.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
```

```
## [1] C/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] caret_6.0-84
##  [2] ggplot2_3.1.1
##  [3] lattice_0.20-38
##  [4] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
##  [5] GenomicFeatures_1.36.0
##  [6] AnnotationDbi_1.46.0
##  [7] Biobase_2.44.0
##  [8] GenomicRanges_1.36.0
##  [9] GenomeInfoDb_1.20.0
## [10] IRanges_2.18.0
## [11] S4Vectors_0.22.0
## [12] BiocGenerics_0.30.0
## [13] transcriptR_1.12.0
## [14] BiocStyle_2.12.0
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-139               bitops_1.0-6
##  [3] matrixStats_0.54.0         lubridate_1.7.4
##  [5] bit64_0.9-7                RColorBrewer_1.1-2
##  [7] progress_1.2.0             httr_1.4.0
##  [9] tools_3.6.0                R6_2.4.0
## [11] rpart_4.1-15               DBI_1.0.0
## [13] lazyeval_0.2.2             colorspace_1.4-1
## [15] nnet_7.3-12                withr_2.1.2
## [17] tidyselect_0.2.5           prettyunits_1.0.2
## [19] bit_1.1-14                 compiler_3.6.0
## [21] DelayedArray_0.10.0        labeling_0.3
## [23] rtracklayer_1.43.3         bookdown_0.9
## [25] scales_1.0.0               stringr_1.4.0
## [27] digest_0.6.18              Rsamtools_2.0.0
## [29] rmarkdown_1.12             XVector_0.24.0
## [31] pkgconfig_2.0.2            htmltools_0.3.6
## [33] highr_0.8                  rlang_0.3.4
## [35] RSQLite_2.1.1              generics_0.0.2
## [37] hwriter_1.3.2              BiocParallel_1.18.0
## [39] dplyr_0.8.0.1              ModelMetrics_1.2.2
## [41] RCurl_1.95-4.12            magrittr_1.5
## [43] GenomeInfoDbData_1.2.1     Matrix_1.2-17
## [45] Rcpp_1.0.1                 munsell_0.5.0
## [47] pROC_1.14.0                stringi_1.4.3
## [49] yaml_2.2.0                 MASS_7.3-51.4
## [51] SummarizedExperiment_1.14.0 zlibbioc_1.30.0
## [53] plyr_1.8.4                 recipes_0.1.5
## [55] grid_3.6.0                 blob_1.1.1
```

```
## [57] crayon_1.3.4                Biostrings_2.52.0
## [59] splines_3.6.0               hms_0.4.2
## [61] knitr_1.22                  pillar_1.3.1
## [63] reshape2_1.4.3              codetools_0.2-16
## [65] biomaRt_2.40.0              XML_3.98-1.19
## [67] glue_1.3.1                  ShortRead_1.42.0
## [69] evaluate_0.13               latticeExtra_0.6-28
## [71] data.table_1.12.2           BiocManager_1.30.4
## [73] foreach_1.4.4               gtable_0.3.0
## [75] purrr_0.3.2                 assertthat_0.2.1
## [77] chipseq_1.34.0              xfun_0.6
## [79] gower_0.2.0                 prodlim_2018.04.18
## [81] e1071_1.7-1                 class_7.3-15
## [83] survival_2.44-1.1           timeDate_3043.102
## [85] tibble_2.1.1                iterators_1.0.10
## [87] GenomicAlignments_1.20.0    memoise_1.1.0
## [89] lava_1.6.5                  ipred_0.9-9
```

7 References