# WhistleR: extract genomic features for predictive modeling in genomics

***Zhen Wei***[*1,2]***, Daiyun Huang***[†1,3]***, Yu Zhong***[‡1,4]***, and Jia Meng***[§1,2]

[1]Department of Biological Sciences, Xi'an Jiaotong-Liverpool University, Suzhou, Jiangsu, China
[2]Institute of Integrative Biology, University of Liverpool, Liverpool, United Kingdom
[3]Department of Computer Science, University of Liverpool, Liverpool, United Kingdom
[4]Graduate Program in Bioinformatics, Boston University, Boston, United States of America

[*]`Zhen.Wei@liverpool.ac.uk` [†]`Daiyun.Huang@liverpool.ac.uk` [‡]`Yu.Zhong@xjtlu.edu.cn`
[§]`Jia.Meng@liverpool.ac.uk`

**2021-03-03**

## Contents

1 Introduction . . . 2
1.1 Overview of functionalities . . . 2
1.2 Installation . . . 4
1.3 Standard workflow . . . 5
2 Make Regions from New Sources . . . 8
2.1 Utilize the Ensembl database . . . 8
2.2 Self-defined regions to extract properties . . . 8
3 More Feature Engineering Options . . . 9
3.1 Design novel genome-derived region properties . . . 9
3.2 Extracting topologies on transcript . . . 11
3.3 Other features from transcript databases . . . 12
4 Case Studies . . . 13
4.1 Setup and EDA . . . 13
4.2 Train machine learning models with h2o package . . . 16
4.3 High performance modeling with AutoML . . . 17
5 References . . . 21

# 1 Introduction

The guide will offer an overview on the work flow of Bioconductor package WhistleR to annotate comprehensive genome-derived features on the range-based genomic objects. The package can be used to extract properties of region lengths, sequence contents, relative positions, clustering effects, and conservation scores on top of the classically defined gene annotations including 5'UTR, CDS, 3'UTR, transcripts, genes, and promoters. The genome-derived features can be used to predict variety of gene related genomic assays, such as the RBP binding sites and the RNA modification sites. WhistleR also provide the annotation of new attributes over user-defined genomic regions, enabling the augmentation of feature engineering under the interactive framework between genomic properties and genomic regions.

The package currently implements 2 types of feature extraction modules: the **genome-derived features** and the **sequence-derived features**.The former is encoded via the interaction between genomic properties and the genomic regions, and the later is defined through the nucleotide encoding methods, such as the one-hot encoding or the encoding of pseudo nucleotide compositions (PseTNC)[1]. The genome and sequence features can be extracted with the function "*genomeDerivedFeatures*" and "*sequenceDerivedFeatures*" respectively. Furthermore, more genome-derived features can be defined using functions with names "`extractRegion`" + the metric names. While the individual extractor functions can annotate new properties on the user defined region types.

The design of package enables the one-step extraction of hundreds of genomic features based only on the *GRanges* of the target genome intervals. Wide class of the Bioconductor annotation objects, including *Txdb*, *EnsDb*, *BSgenome*, and *GScores*, are supported in the package to enable the massive enumeration of features that based on novel sources of genome regions.

While the comprehensive feature input is vital to the success of a genomic data science project, the genome-derived features are often more informative and interpretive than the primary sequence features. Thus, the *WhistleR* package can provide general support for machine learning modeling in the domain of genomics.

The user's guide will firstly give an overview to the key functionalities of the package, then it will illustrate the utility of the predictive features using a case study of the full workflow in a genomic prediction task, from EDA to modeling.

## 1.1 Overview of functionalities

*WhistleR* provides fast and comprehensive feature extraction from the interval-based genomic data; the extracted features can be used as the input for a variety of modeling purposes such as supervised predictive modeling and unsupervised factor analysis. The target range-based annotation should be stored in *GRanges* object, which can be created using the constructor function defined in the *GenomicFeatures* package. Alternatively, the *GRanges* can be imported from the external annotation files in **BED**, **GFF**, or **GTF** formats using the `import` function defined in *dplyr* package.

There are 2 types of genomic features implemented in *WhistleR*: the region properties features, and the sequence-derived features. The former is extracted using an interactive design between the genomic properties and the genomic regions, the latter is the sequence-derived features extracted from the sequences of the flanking regions of the target. When extracting the sequence-derived features, different encoding methods can be specified, such as the one-hot method or the pseudo nucleotide composition method.

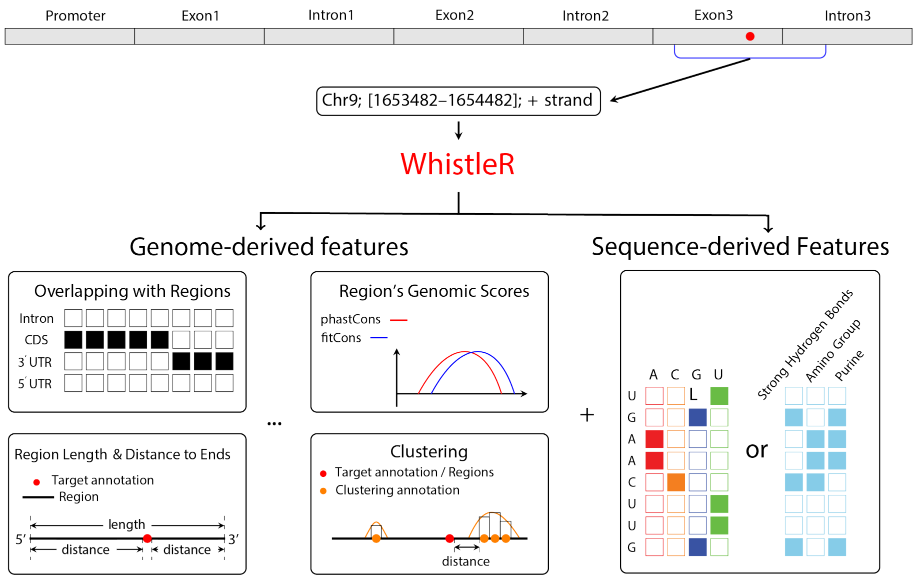


**Figure 1:** Feature extraction framework of WhistleR

### 1.1.1 Genome-derived features encoding

The genome-derived features are encoded as the interaction between the genomic regions (e.x. regions extracted from the gene annotations: 5'UTR, CDS, 3'UTR) and the genomic properties over genomic-intervals (e.x length, GC content), so we name this feature encoding method **region properties features**. Formally, the region properties features is a matrix $\mathbf{X}_{RP}$ defined by:

$$\mathbf{X}_{RP} = (\mathbf{X}_R, \mathbf{X}_P, \mathbf{X}_{R\times P})$$

Let $n$ denotes the total number of interval-based target annotations, while $d$ and $g$ are the number of regions and properties for the feature extraction. $\mathbf{X}_R$ is an $n \times d$ matrix, where it columns are indicator variables for the range $i$ overlaps with the region $j$.

$$\mathbf{X}_R^{(i,j)} = \mathbf{1}(c_A^{(i)} \cap c_R^{(j)} \neq \emptyset)$$

The $c_A^{(i)}$ is the genome coordinate values spanned by the annotation $i$, while the $c_R^{(j)}$ is the genome coordinate values spanned by the region $j$. $X_P$ is an $n \times g$ dimensional matrix, and each column of $X_P$ is a vector of the properties $k$ extracted from the target annotation:

$$\mathbf{X}_P^{(i,k)} = \mathbf{prop}^{(k)}(c_A^{(i)})$$

$\mathbf{prop}^{(k)}$ is a function that can map a set of genome coordinates to a numeric value of property $k$. Instances of such functions can be the length of the interval and the GC content of the interval.

Next, we add a set of interactive features $\mathbf{X}_{R\times P}^{(i,h)}$ between the indicator variables and the genomic properties. It represents an $n \times (d + g)$ matrix defined as:

$$\mathbf{X}_{R\times P}^{(i,h)} := \mathbf{X}_R^{(i,j)} \times \mathbf{prop}^{(k)}(c_R^{(i,j)})$$

$c_R^{(i,j)}$ is the coordinates for the subset of genomic ranges in region $j$ that overlapped with the target annotation $i$. While $h$ is indexing the columns of the interaction matrix, and $h = j + d(k-1)$. The calculation for the overlapping relationships between the target annotations and regions is realized through the `findOverlaps` method exported from the package *GenomicRanges*.

### 1.1.2 Sequence-derived feature encoding

The one-hot encoded sequence feature matrix $X_{onehot}$ is represented by the following equation; the $i, j$ are indexing the $i$th annotation and the $j$th nucleotide position within the flanked DNA sequence.

$$
\begin{aligned}
X_{onehot}^{(i,(j-1)*4+1)} &= \mathbf{1}(s_i^{(j)} = \text{A}) \\
X_{onehot}^{(i,(j-1)*4+2)} &= \mathbf{1}(s_i^{(j)} = \text{T}) \\
X_{onehot}^{(i,(j-1)*4+3)} &= \mathbf{1}(s_i^{(j)} = \text{C}) \\
X_{onehot}^{(i,(j-1)*4+4)} &= \mathbf{1}(s_i^{(j)} = \text{G})
\end{aligned}
$$

$s_i^{(j)}$ is the nucleotide $j$ in the sequence of the $i$th annotation. In *WhistleR*, the input ranges for the sequence-derived feature extraction must have the same width.

Often, compared with conventional one-hot encoding approach, more sophisticated sequence encoding methods involving the physio-chemical properties of the nucleotides can improve the predictive performance. *WhistleR* implemented the pseudo-neocleotide composition (PseTNC) encoding frequently used in genome prediction projects. The feature matrix $X_{PseTNC}$ of PseTNC is defined as follows:

$$
\begin{aligned}
X_{PseTNC}^{(i,(j-1)*4+1)} &= \mathbf{1}(s_i^{(j)} \in \{\text{A,G}\}) \\
X_{PseTNC}^{(i,(j-1)*4+2)} &= \mathbf{1}(s_i^{(j)} \in \{\text{A,C}\}) \\
X_{PseTNC}^{(i,(j-1)*4+3)} &= \mathbf{1}(s_i^{(j)} \in \{\text{A,T}\}) \\
X_{PseTNC}^{(i,(j-1)*4+4)} &= \frac{1}{j} \sum_{k=1}^{j} \mathbf{1}(s_i^{(k)} = s_i^{(j)})
\end{aligned}
$$

The first three features of PseTNC are indicators of the three physio-chemical properties of nucleic acids; those are having 2 rings structure (A,G), having ammino groups (A,C), and forming weak hydrogen bonds with its complementary bases (A,T), respectively. The fourth feature is the cumulative frequency of nucleotides from the leftmost position to the position $j$. Both the one-hot encoding and PseTNC encoding will generate a feature matrix with dimension $n \times 4l$, and $l$ is the length of each sequence.

## 1.2 Installation

To install *WhistleR* from bioconductor, start R (version >"4.2") and enter:

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

For order versions of R, please refer to the appropriate Bioconductor release.

## 1.3 Standard workflow

### 1.3.1 Reading interval-based annotation from a file

If the target annotation is stored in a *BED*, *GTF*, or *GFF* file, it can be easily loaded into R using the `import()` function in the *rtracklayer* package. The format of the file can be obtained directly from the file extension or manually specified by the `format` argument. The output of `import()` is a *GRanges* object, so it can be directly used as the input of the feature extraction function.

```
## Load the example bed annotation file into R with import function
bed_dir <- system.file("extdata", "GSE63753.bed",
                       package = "WhistleR")
X <- rtracklayer::import(bed_dir)
```

When loading annotation files saved in other tabular formats, such as *TXT or CSV* files, the table should be first loaded into R using `read.table()` or `read.csv()`. Then, the *GRanges* can be constructed from the loaded *data.frame* using the function `makeGRangesFromDataFrame()` defined in the *GenomicRanges* package.

The column labeling of the key fields, including the *seqnames* (chromosome info), *start*, *end/width*, and *strand* of the annotation, need to be consistent with the arguments in `makeGRangesFromDataFrame()`. Please see `?makeGRangesFromDataFrame` for the detailed usage if the columns of your *data.frame* are named differently from the default settings.

```
## Create the annotation Granges using GRanges() function
library(GenomicRanges)
tsv_dir <- system.file("extdata", "GSE63753.txt",
                       package = "WhistleR")
X_df <- read.table(tsv_dir, sep = "\t", header = TRUE)
X <- makeGRangesFromDataFrame(X_df)
```

### 1.3.2 Genome-derived features

In a standard workflow of *WhistleR*, we will first use the `genomeDerivedFeatures()` function to extract the genome-derived features. In general, the feature matrix returned by the function is encoded by the region properties encoding method described in 1.1.1, and the corresponding regions and properties are extracted from the target annotation objects. The following parts of this section will explain the usage and the principles of the genome-derived feature extraction.

First, load all the packages required for the feature extraction:

```
library(WhistleR)
library(TxDb.Hsapiens.UCSC.hg19.knownGene) ##Txdb for transcript annotation
library(BSgenome.Hsapiens.UCSC.hg19) ##BSgenome for genome sequence
library(phastCons100way.UCSC.hg19) ##GScores for genomic scores
```

We change the variables of the annotation packages into concise names:

```
txdb_hg19 <- TxDb.Hsapiens.UCSC.hg19.knownGene
genome_hg19 <- BSgenome.Hsapiens.UCSC.hg19
phastCons_hg19 <- phastCons100way.UCSC.hg19
```

Next, extract of the genome-derived features on the target GRanges object. The detailed structure of each column can be displayed by the `str()` function (not evaluated in the document).

```
RPF <- genomeDerivedFeatures(X,
                             ## Providing gene annotation
                             transcriptdb=txdb_hg19,
                             ## Providing genome sequence
                             sequence=genome_hg19,
                             ## Providing Phastcons scores
                             gscores=phastCons_hg19,
                             ## Calculate clustering metrics on X itself
                             clusteringY=X)
## Display the outcomes of feature extraction
str(RPF)
```

Table1 summarizes the default genomic regions extracted from the transcript annotation specified by `transcriptdb`:

**Table 1:** Default genomic regions extracted from transcriptdb

| Region | Description |
|---|---|
| Exons | Exons defined in transcriptdb. |
| Introns | Intronic parts defined in transcriptdb. |
| Exonic 5'UTR | The exonic parts of 5'UTR. |
| Full 5'UTR | Full 5'UTR (with introns). |
| Exonic CDS | The exonic parts of CDS. |
| Full CDS | Full CDS (with introns). |
| Exonic 3'UTR | The exonic parts of 3'UTR. |
| Full 3'UTR | Full 3'UTR (with introns). |
| Exonic Transcript | Mature RNA transcript. |
| Full Transcript | Full transcript (with introns). |
| Exonic Genes | The exonic parts of genes. |
| Full Genes | Full genes (with introns). |
| Promoters | Promoter regions of transcripts. |

Subsequently, Table 2 lists the properties calculated from the genomic regions defined above.

**Table 2:** Properties calculated from the genomic regions

| Property | Description |
|---|---|
| Length | Region length. |
| Sequence content | Sequence content of the region (default GC content). |
| Genomic scores | Average genomic scores of the region. |
| Count of Y | Count of the overlapped annotation Y on region. |
| Density of Y | Density of the overlapped annotation Y on region. |
| Nearest distance to Y | Nearest distance from the region to annotation Y. |
| Relative position | Relative position of X on the region. |
| Distance to 5'end | Distance of X to region's 5'end. |
| Distance to 3'end | Distance of X to region's 3'end. |

All of the gene regions listed in Table 1 are computed using the functions defined in the package `GenomicFeatures`. By default, the promoters obtained are the upstream 2000bp and downstream 200bp of the transcription start sites (TSS).

The density of $Y$ (The clustering genomic annotation) on the region is defined as:

$$\text{Density of Y on region} = \frac{\text{Count of Y on region}}{\text{Region length}}$$

The relative position of $X$ on the region is defined as:

$$\text{Relative position of X on region} = \frac{\text{Distance of X toward region's 5'end}}{\text{Region length}}$$

The sequence content calculated by default is the GC content. Under the argument setting used in the above example, the genomic scores used is the PhastCons scores of hg19.

In the example above, we used $\text{Y} = \text{X}$ to calculate the properties related to the clustering effect on the genome coordinate, including the count of Y, the density of Y, and the nearest distance to Y. That is, in our case, these properties are used to quantify the self clustering potential of the target annotation.

Finally, other than the properties described in Table 2, the function will also generate 3 additional features to describe the unique biological properties of the region type of exonic genes: gene's exon number, gene's transcript isoform number, and Meta-TX topology. Among them, the Meta-TX topology is calculated by the following equation:

$$\text{Meta-tx topology} = \frac{1}{3} \times (\text{Pos(5'utr)}\mathbf{1}(\text{5'utr}) + (1 + \text{Pos(cds)})\mathbf{1}(\text{cds}) + (2 + \text{Pos(3'utr)})\mathbf{1}(\text{3'utr}))$$

Pos(region) is the relative position of the target in the corresponding region (as defined above); $\mathbf{1}(\text{region})$ is an indicator function for the target overlaps the region. $\mathbf{1}(\text{region}) = 1$ if the target overlaps the region, otherwise $\mathbf{1}(\text{region}) = 0$. The Meta-TX topology is a simplified version of the transcript distribution described in the guitar package [2] and the Meta-TX method [3].

### 1.3.3 Sequence-derived features

The information derived from nucleotide sequences are often the most fundamental features when building prediction models for the genome annotations, and it is recommended to combine both the sequence and genome derived features to achieve the highest model performance in any genomic data science projects.

*WhistleR* implemented different sequence encoding methods in function `sequenceDerivedFeatures()`; the function can extract the features under the given encoding schema directly from the genomic intervals defined in GRanges object. In order to generate a complete feature matrix in the output, the input GRanges of `sequenceDerivedFeatures()` are required have the same width/length. Therefore, when extracting sequence features, it is recommended to use the `resize()` function to fix the size of the annotation object.

We first define the range of sequence extraction as 41bp windows centered on the target annotation. Since the ranges in the original annotation are single based modification sites, resizing is equivalent to adding 20bp flanking regions on both sides of the sites.

```
X_resized <- resize(X, 41, fix = "center")
```

The sequence features of One-hot encoding can be extracted with:

```
Onehot_SF <- sequenceDerivedFeatures(X_resized,
                                     sequence=genome_hg19,
                                     encoding = "onehot")
str(Onehot_SF)
```

Alternatively, extract the sequence features of PseTNC encoding with:

```
PseTNC_SF <- sequenceDerivedFeatures(X_resized,
                                     sequence=genome_hg19,
                                     encoding = "iRNA")
str(PseTNC_SF)
```

# 2 Make Regions from New Sources

The genomic regions used when extracting genome-derived features can be replaced and expanded to further enlarge the feature space explored by the region properties encoding. For example, since the number of transcripts contained in the Ensemble database is more than twice that of UCSC, we can replace *TxDb* with *ensembldb* in the transcript annotation input.

## 2.1 Utilize the Ensembl database

To work with transcript annotation from Ensembl, we first need to install/load the EnsDb data package corresponding to the genome version of the target annotation. The target in the example comes from hg19, so the v75 of the Human annotation on Ensembl will be used.

```
library(EnsDb.Hsapiens.v75)
ensdb_hg19 <- EnsDb.Hsapiens.v75
```

The loaded *ensembldb* package can directly substitute the *TxDb* package at the `transcriptdb` argument:

```
## Extract region properties using the annotation of Ensemble database
RPF <- genomeDerivedFeatures(X,
                             transcriptdb=ensdb_hg19,
                             sequence=genome_hg19,
                             gscores=phastCons_hg19,
                             clusteringY=X)

str(RPF)
```

Now, the region properties features returned are based on the transcriptomic regions from Ensembl database.

## 2.2 Self-defined regions to extract properties

All of the above examples operate on the 13 basic types of gene regions (listed on Table 2). However, depending on the biological nature of the target of interest, genomic regions other than gene/transcript annotations may contain critical information for modeling. For example, some RNA binding proteins realize their functions through regulating the RNA secondary structures [7], and epigenetic modifications often directly regulate the chromosome structures [8]. In case of constructing high-performance predictors on these targets, additional regions can be defined using the interval based annotations of RNA secondary structures and chromosomal conformations.

*WhistleR* support the extraction of properties over an arbitrary set of genomic regions defined in the *GRanges* or *GRangesList* class. The example below adds tRNA on h19 as the additional region for the feature extraction. The GRanges for tRNA can be extracted from the Txdb object of hg19.

```
tRNAs_hg19 <- tRNAs(txdb_hg19)
```

Next, we provide the additional annotation at the argument `extraRegions`:

```
RPF <- genomeDerivedFeatures(X,
                             transcriptdb=txdb_hg19,
                             sequence=genome_hg19,
                             gscores=phastCons_hg19,
                             clusteringY=X,
                             extraRegions=list(tRNA=tRNAs_hg19))

str(RPF)
```

Then, the tRNA related features will be extracted in additional to the 14 basic region types, and all properties described in 1.1.1 will be applied to the newly provided region. Please also note that the input for `extraRegions` can be a *list* of *GRanges* or *GRangesList*. The region type features will be interactively retrieved for each element of the list. Meanwhile, the extracted new features will be labeled, as shown in the column headers of the returned table, by the **names** of the list.

# 3 More Feature Engineering Options

*WhistleR* offers separate feature extraction functions for end-users who want to deign their own region properties. In general, these functions enable the extraction of one feature at a time given a region object and a specific metric of interest.

## 3.1 Design novel genome-derived region properties

The example below demonstrates the extraction of single genome-derived features from a randomly sampled query *GRanges* on the exons of hg19 genome.

Firstly, we will build the query *GRanges* object using:

```
library(GenomicRanges)
x_gr <- GRanges(rep(c("chr1", "chr2"), c(5, 15)),
                IRanges(c(sample(11874:12127, 5),
                          sample(38814:41527,15)), width=100),
                strand=Rle(c("+", "-"), c(5, 15)))
x_gr
```

Next, we manually specify the 3 exons of human genome as region object, and then re-structure the *GRanges* into *GRangesList* to create a list of 2 elements representing 2 genes:

```
exons_gr <- GRanges(c("chr1","chr2","chr2"),
                    IRanges(start=c(11874,38814,45440),
                            end=c(12227,41627,46588)),
                     strand=c("+","-","-"))
genes_grl <- GRangesList(gene1=exons_gr[1],gene2=exons_gr[c(2,3)])
```

Then, we use the function extractRegionLength along with the query to extract the lengths of GRanges:

```
extractRegionLength(x_gr)
```

When adding `region=exon_gr`, the length of the regions (exons) defined in the region will be extracted, and the mapping is determined by the overlap relationships between query and exons.

```
extractRegionLength(x_gr, region=exons_gr)
```

Similarly, the region can be a `GRangesList`, in which the following case will extract the exonic length of the genes overlapped by the query.

```
extractRegionLength(x_gr, region=genes_grl)
```

Self defined properties can be specified as a numeric vector with the same length of the region object, and the returned value is a vector that map the properties value from the region to the query.

```
exons_property <- c(1,6,8)
extractRegionProperty(x_gr, region=exons_gr, property=exons_property)
```

Furthermore, WhistleR defined property extractors that act on the genome sequence and genome score packages.

```
library(BSgenome.Hsapiens.UCSC.hg19)
bsgenome <- BSgenome.Hsapiens.UCSC.hg19
```

`extractRegionLetterFrequency()` can be used to access to GC content of the query GRanges.

```
extractRegionLetterFrequency(x_gr,
                             sequence=bsgenome,
                             letters="GC")
```

By passing different DNA string to the letters argument, one can calculate other sequence content (e.x. "A content") from x.

```
extractRegionLetterFrequency(x_gr,
                             region=exons_gr,
                             sequence=bsgenome,
                             letters="A")
```

After providing the argument region, the sequence contents are now calculated for the indiviual regions, and then mapped to the corresponding query overlapped.

```
extractRegionLetterFrequency(x_gr,
                             region=exons_gr,
                             sequence=bsgenome,
                             letters="GC")
```

The individual properties extractors constitute the fundamental building blocks of the region properties features extracted by the one-step function. The major utility of the individual functions is to enable users to annotate their newly defined properties with the function `extractRegionProperty()`. In the later versions of WhistleR, more genomic metrics will be constructed to enrich the set of properties available for RPF.

## 3.2 Extracting topologies on transcript

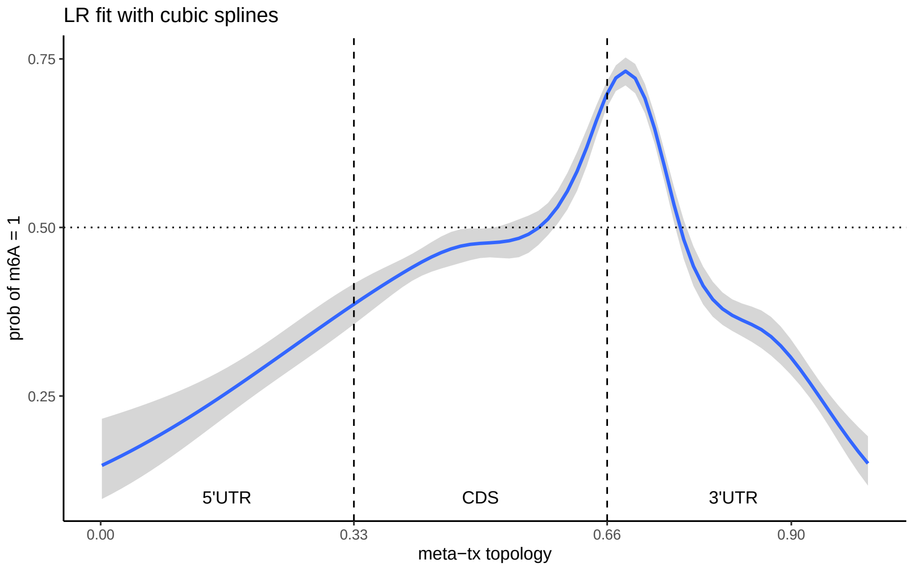
The distribution of genomic markers on transcript coordinate (Travis coordinate) can often provide insight into its biological functions. Usually, such distribution plot along is drawn by an individual package (such as the Guitar bioconductor package). The relative position on transcript coordinate can be highly important for the modeling of RNA related molecular targets such as the m6A modification on messenger RNA [5].

*WhistleR* export a single function `topologyOnTranscripts()` to provide a fast extraction of transcript topology from query GRanges annotation. The required input is the *GRanges* and the corresponding *TxDb* object. The following example uses the annotation of the m6A miCLIP dataset; while positive sites are single based m6A modification sites (restricted on DRACH motif), and the negative sites are randomly sampled DRACH motifs from the exonic transcripts containing the positive sites.

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene

## Load the exemplar annotation of m6A miCLIP dataset
GSE63753_sysy <- readRDS(system.file("extdata", "GSE63753_sysy.rds",
                                     package = "WhistleR"))

GSE63753_sysy$topology <- topologyOnTranscripts(GSE63753_sysy, txdb)
```

Next, after extracting the topology of the data set on transcript, we can plot the relationship between the meta-tx toplogy and the binary target using logistic regression. Such plot can be quickly implemented with the `geom_smooth()` function defined in package *ggplot2*. Smoothing splines are applied so that the smoothed logistic regression curve can reflect the non-linear effect of transcript topology on the target outcome (in this case, the m6A modification site).

```
library(ggplot2)
ggplot(na.omit(as.data.frame(mcols(GSE63753_sysy))), aes(topology, target)) +
  geom_smooth(
    formula = y~splines::ns(x, 8),
    method = "glm",
    method.args = list(family = "binomial")
  ) +
  geom_vline(xintercept = c(0.33, 0.66), linetype = 2) +
  geom_hline(yintercept = 0.5, linetype = 3) +
  geom_text(aes(x = x, y = y, label = text),
    data = data.frame(
      x = c(0.165, 0.495, 0.825),
      y = c(0.1, 0.1, 0.1),
      text = c("5'UTR", "CDS", "3'UTR")
    )) + scale_x_continuous(breaks = c(0, 0.33, 0.66, 0.9)) +
         scale_y_continuous(breaks = c(0, 0.25, 0.5, 0.75, 1)) +
  theme_classic() + labs(x = "meta-tx topology",
                         y = "prob of m6A = 1",
                         title = "LR fit with cubic splines")
```



Please note that the interpretation plot generated by this approach only demonstrates the marginal effect of genome-derived features on the prediction of its target. In practice, modeling is usually performed with more than one feature, and its prediction may rely on the interactions between multiple features. However, such single feature LR curve is still useful for EDA, and the pattern of non-linear effects may gain useful insight for the genome-derived feature in relation to the target.

## 3.3 Other features from transcript databases

The transcript databases often contain other biologically important information that is not limited to the region$*$properties format, examples of these features include chromosomal information and the gene/transcript biotypes. *WhistleR* allows to extract those annotations that beyond the encoding schema of the region properties features.

If the *EnsDb* object is provided in the `transcriptdb` of `genomeDerivedFeatures()`, the **transcript biotype** included in the ensembleDb can be automatically retrieved as the additional columns of the feature matrix when adding `annotBiotype=TRUE`. In addition, by setting the `annotSeqnames=TRUE`, the function will further extract the *seqnames* of the targets which mapped to the *seqlevels* of the provided *transcriptdb*.

```
library(EnsDb.Hsapiens.v75)
## Generate additional features of seqnames and transcript biotype
RPF <- genomeDerivedFeatures(X,
                             transcriptdb=EnsDb.Hsapiens.v75,
                             sequence=BSgenome.Hsapiens.UCSC.hg19,
                             gscores=phastCons100way.UCSC.hg19,
                             clusteringY=X,
                             annotSeqnames=TRUE,
                             annotBiotype=TRUE)
```

# 4 Case Studies

The case introduced below use the miCLIP data generated in Linder *et al*. [4]. The single based resolution m6A sites are publicly available from the Gene Expression Omnibus (GEO) at the accession number GSE63753. The study examines the base resolution m6A modification sites in human and mouse with different antibodies and site calling methods. Here, we attempt to build a prediction model to classify the human m6A modification (identified using the Abcam antibody) from a set of negative DRACH sites. The positive sites are filtered by the miCLIP sites that mapped to the exons and the DRACH motif, and the negative sites are randomly sampled unmethylated DRACH motifs from the exonic regions of the transcripts that contain the positive data. The number of negative sampled is equal to the number of positives so that the classification data set is balanced.

## 4.1 Setup and EDA

We first load the packages for sequence, transcript annotation, and the genomic conservation scores on hg19:

```
library(WhistleR)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(BSgenome.Hsapiens.UCSC.hg19)
library(phastCons100way.UCSC.hg19)

txdb_hg19 <- TxDb.Hsapiens.UCSC.hg19.knownGene
genome_hg19 <- BSgenome.Hsapiens.UCSC.hg19
phastCons_hg19 <- phastCons100way.UCSC.hg19
```

Next, load the GRanges of the m6A miCLIP data prepared for classification modeling:

```
GSE63753_abcam <- readRDS(system.file("extdata",
                          "GSE63753_abcam.rds",
                          package = "WhistleR"))

GSE63753_abcam
## GRanges object with 14740 ranges and 1 metadata column:
##           seqnames    ranges strand |    target
##              <Rle> <IRanges>  <Rle> | <numeric>
##              chr10 100176312      - |         1
##              chr10 101090592      + |         1
##              chr10 101503021      + |         1
##              chr10 101515461      + |         1
##              chr10 101515594      + |         1
##       ...      ...       ...    ... .       ...
##      chr2     chr2  55237442      - |         0
##     chr11    chr11  46832674      - |         0
##    chr5     chr5 167919818      + |          0
##    chr6     chr6  89322456      - |          0
##    chr4     chr4  24557939      - |          0
##   -------
##   seqinfo: 93 sequences from an unspecified genome
table(GSE63753_abcam$target)
##
##    0    1
## 7370 7370
```

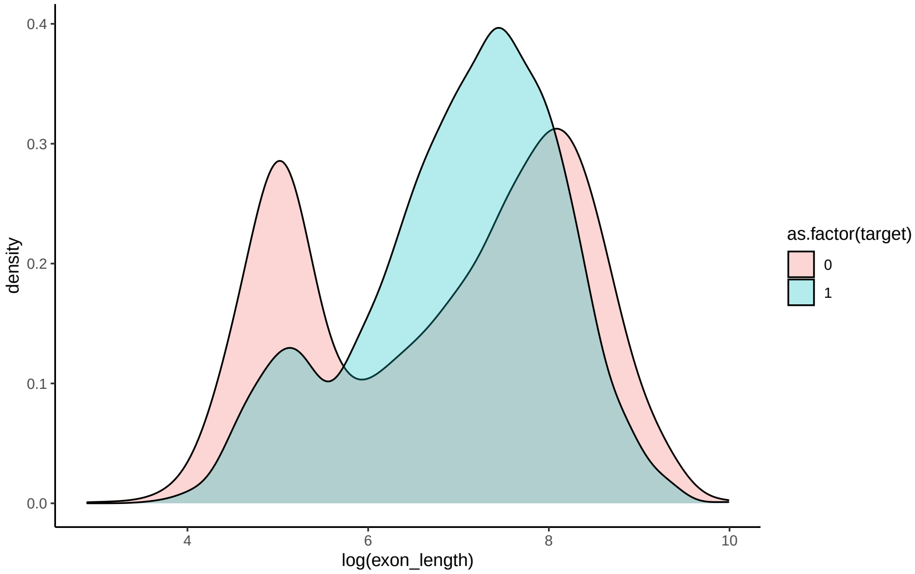
The metadata column of the GRanges is a dummy (0/1) vector, 1 is the positive m6A site, and 0 is the negative DRACH motif. The negative data is randomly sampled from the regions of positive containing transcript, and the negative sampling function used is `sampleSequence()`, which can sample ranges of a given sequence motif from any region of a genome. Please, see `?sampleSequence()` for more illustrations on this task, as the correct sampling of negative data is often crucial for the success of a data science project in genomics.

Prior to the feature extraction and model training process, we could first examine the relationship between exon length and m6A modification, which is previously reported by numerous studies [5,6].

```
## Extract exon length overlapped by the DRACH sites:
GSE63753_abcam$exon_length <- extractRegionLength(GSE63753_abcam,
                                                  exons(txdb_hg19))
```

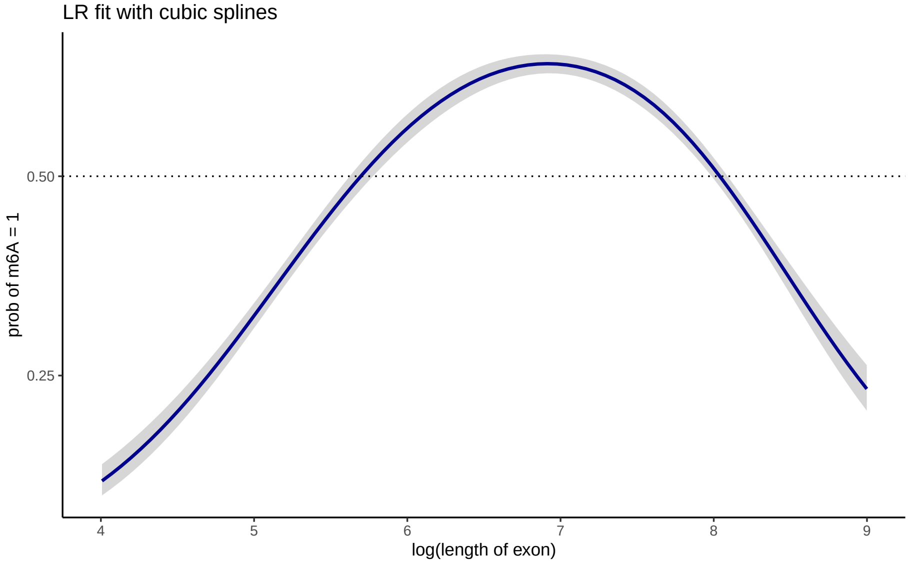
We will first visualize the joint distribution by plotting the densities of exon length stratified by the m6A labeling:

```
library(ggplot2)
plot_df <- na.omit(as.data.frame(mcols(GSE63753_abcam)))

## Plot the distribution of exon length conditioned by labels:
ggplot(plot_df) +
  geom_density(aes(x=log(exon_length),
                   fill = as.factor(target)),
               alpha = 0.3) + theme_classic()
```



Next, we will visualize the conditional distribution by fitting the smoothed logistic regression using the exon length as covariate:

```
## Plot the logistic regression fit with cubic splines:
ggplot(plot_df, aes(log(exon_length), target)) +
  geom_smooth(formula = y ~ splines::ns(x, 3),
              method = "glm", method.args = list(family = "binomial"),
              color = "dark blue") +
  geom_hline(yintercept = 0.5, linetype = 3) +
  scale_x_continuous(limits = c(4,9)) +
  scale_y_continuous(breaks = c(0, 0.25, 0.5, 0.75, 1)) +
  theme_classic() + labs(x = "log(length of exon)",
                         y = "prob of m6A = 1",
                         title = "LR fit with cubic splines")
```



With the help of B-splines, we could clearly see that the effect of exon length on the probability of the m6A modification is not linear. In other words, although the m6A is more likely to be seen in the longer exons. However, if the exon length is too long, it becomes less probable to be methylated compared to the medium-length exons.

Following that, the region properties features and the sequence features are extracted from the target annotation. This time, the clustering related properties are calculated with all DRACH motifs on the exonic regions of hg19. The inputs for *gscores* and *clusteringY* are now wrapped in a `list` in order to label the corresponding features with the names of the list elements.

```
## Retrieve all DRACH motif on the exon regions of hg19
exons_DRACH <- sampleSequence("DRACH", exons(txdb_hg19), genome_hg19)

RPF <- genomeDerivedFeatures(GSE63753_abcam,
                             transcriptdb=txdb_hg19,
                             sequence=genome_hg19,
                             gscores=list(phastCons=phastCons_hg19),
                             clusteringY=list(motif=exons_DRACH))

PseTNC <- sequenceDerivedFeatures(resize(GSE63753_abcam, 41, fix="center"),
                                  ## Providing the genome sequence
                                  sequence=genome_hg19,
                                  encoding = "iRNA")
```

Then, we will concatenate the 2 feature matrices and use h2o to build the prediction models.

## 4.2 Train machine learning models with h2o package

*h2o* R package is an interface of the open source machine learning platform h2o, which offers implementations of many supervised machine learning algorithms as well as a fully automatic machine learning algorithm (h2o AutoML).

The model matrix we use is combined from the region property features and the PseTNC encoded sequence features extracted above.

```
Model_matrix <- cbind(RPF,PseTNC)
Model_matrix$target <- as.factor(GSE63753_abcam$target)
```

Call *h2o.init()* first to launch h2o on the local computer, and then use *h2o.xgboost()* to train the classification model using the XGBoost (eXtreme Gradient Boosting) algorithm.

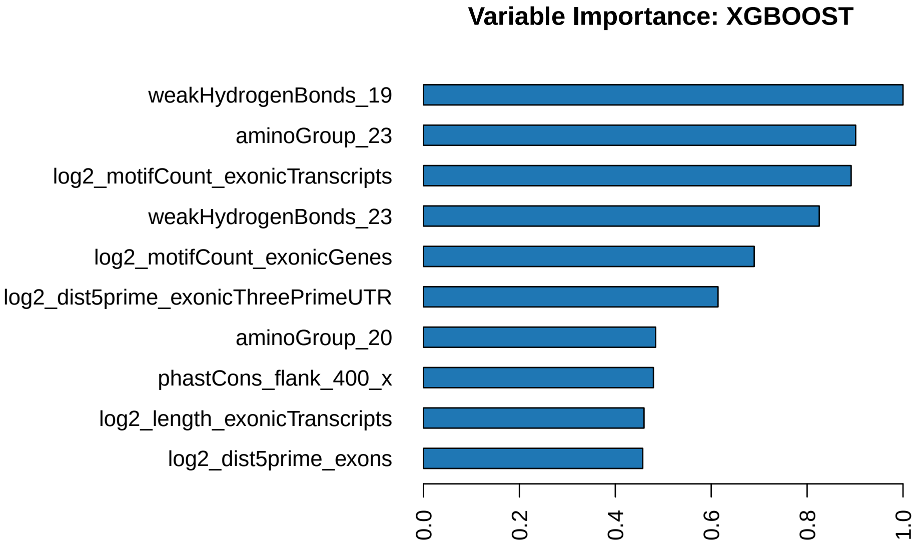
```
library(h2o)
h2o.init()
predictors <- setdiff(colnames(Model_matrix), "target")

## Train the XGB model with 5 folds cross validation
model_xgb <- h2o.xgboost(x = predictors, y = "target",
                           training_frame = as.h2o(Model_matrix),
                           nfolds = 5,
                           learn_rate = 0.05,
                           gamma = 5,
                           max_depth = 5,
                           reg_alpha = 10,
                           reg_lambda = 10,
                           colsample_bytree = 0.3,
                           booster = "gbtree",
                           normalize_type = "tree",
                           seed = 1234)
```

Next, we will check the model performance metric of AUROC on the validation sets:

```
h2o.auc(model_xgb, xval=TRUE) #Check the AUROC using 5 folds cross validation
## [1] 0.8460442
```

The variable importance of the XGBoost model can be calculated and visualized by a bar plot:

```
h2o.varimp_plot(model_xgb)
```



## 4.3 High performance modeling with AutoML

The h2o Automatic Machine Learning (AutoML) function automates the supervised machine learning model training process. Its current version of AutoML trains and cross-validates a Random Forest, an Extremely-Randomized Forest, a random grid of Gradient Boosting Machines (GBMs), a random grid of Deep Neural Nets, and then trains a Stacked Ensemble using all of the models.

We will run AutoML for all the 20 base models (limited to 1 hour max runtime by default):

```
aml <- h2o.automl(x = predictors,
                  y = "target",
                  training_frame = as.h2o(Model_matrix),
                  max_models = 20,
                  nfolds = 5,
                  seed = 1234)
```

Finally, check the leaderboard and performance metrics of the h2o AutoML modeling:

```
lb <- h2o.get_leaderboard(aml)
lb
##                                              model_id       auc   logloss
## 1    StackedEnsemble_AllModels_AutoML_20210129_150031 0.8746530 0.4461841
## 2 StackedEnsemble_BestOfFamily_AutoML_20210129_150031 0.8736283 0.4472846
## 3          GBM_grid__1_AutoML_20210129_150031_model_1 0.8667584 0.4586501
## 4                        GLM_1_AutoML_20210129_150031 0.8651682 0.4592309
## 5                        GBM_5_AutoML_20210129_150031 0.8648396 0.4589557
## 6                        GBM_1_AutoML_20210129_150031 0.8644147 0.4598685
##       aucpr mean_per_class_error      rmse       mse
## 1 0.8652338            0.2020353 0.3785906 0.1433309
## 2 0.8642393            0.2069878 0.3792791 0.1438526
## 3 0.8563012            0.2166893 0.3851365 0.1483301
## 4 0.8566820            0.2147218 0.3857522 0.1488048
## 5 0.8544123            0.2168928 0.3856320 0.1487121
## 6 0.8545153            0.2185210 0.3860874 0.1490635
##
## [22 rows x 7 columns]
```

As we could see from the leaderboard, the top model in AutoML significantly outperform the XGBoost model we trained previously using the same set of features. It is recommended to firstly use the feature importances from XGBoost to conduct feature selection, and then use the more advanced machine learning approaches to enhance the model performance. Similarly, the performance of AutoML is a good criteria for evaluating the overall effectiveness of the feature design for a given task; one can focus on the feature engineering to improve the performance of AutoML without worrying about the selection of hyper-parameters.

The modeling was performed on:

```
## - Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 4.0.2 (2020-06-22)
##  os       macOS Catalina 10.15.4
##  system   x86_64, darwin17.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  ctype    en_US.UTF-8
##  tz       Asia/Shanghai
##  date     2021-03-03
##
## - Packages -------------------------------------------------------------------
##  package                           * version  date       lib source
##  AnnotationDbi                     * 1.50.1   2020-06-29 [1] Bioconductor
##  AnnotationFilter                    1.12.0   2020-04-27 [1] Bioconductor
##  AnnotationHub                       2.20.0   2020-04-27 [1] Bioconductor
##  askpass                             1.1      2019-01-13 [1] CRAN (R 4.0.0)
##  assertthat                          0.2.1    2019-03-21 [1] CRAN (R 4.0.0)
##  Biobase                           * 2.48.0   2020-04-27 [1] Bioconductor
##  BiocFileCache                       1.12.0   2020-04-27 [1] Bioconductor
##  BiocGenerics                      * 0.34.0   2020-04-27 [1] Bioconductor
##  BiocManager                         1.30.10  2019-11-16 [1] CRAN (R 4.0.0)
##  BiocParallel                        1.22.0   2020-04-27 [1] Bioconductor
##  BiocStyle                         * 2.16.0   2020-04-27 [1] Bioconductor
##  BiocVersion                         3.11.1   2020-04-07 [1] Bioconductor
##  biomaRt                             2.44.1   2020-06-17 [1] Bioconductor
##  Biostrings                        * 2.56.0   2020-04-27 [1] Bioconductor
##  bit                                 1.1-15.2 2020-02-10 [1] CRAN (R 4.0.0)
##  bit64                               0.9-7    2017-05-08 [1] CRAN (R 4.0.0)
##  bitops                              1.0-6    2013-08-17 [1] CRAN (R 4.0.0)
##  blob                                1.2.1    2020-01-20 [1] CRAN (R 4.0.0)
##  bookdown                            0.20     2020-06-23 [1] CRAN (R 4.0.2)
##  BSgenome                          * 1.56.0   2020-04-27 [1] Bioconductor
##  BSgenome.Hsapiens.UCSC.hg19       * 1.4.3    2020-07-02 [1] Bioconductor
##  cli                                 2.0.2    2020-02-28 [1] CRAN (R 4.0.0)
##  colorspace                          1.4-1    2019-03-18 [1] CRAN (R 4.0.0)
##  crayon                              1.3.4    2017-09-16 [1] CRAN (R 4.0.0)
##  curl                                4.3      2019-12-02 [1] CRAN (R 4.0.0)
##  DBI                                 1.1.0    2019-12-15 [1] CRAN (R 4.0.0)
##  dbplyr                              1.4.4    2020-05-27 [1] CRAN (R 4.0.0)
##  DelayedArray                        0.14.0   2020-04-27 [1] Bioconductor
##  digest                              0.6.25   2020-02-23 [1] CRAN (R 4.0.0)
##  dplyr                               1.0.0    2020-05-29 [1] CRAN (R 4.0.0)
##  ellipsis                            0.3.1    2020-05-15 [1] CRAN (R 4.0.0)
##  ensembldb                           2.12.1   2020-05-06 [1] Bioconductor
##  evaluate                            0.14     2019-05-28 [1] CRAN (R 4.0.0)
##  fansi                               0.4.1    2020-01-08 [1] CRAN (R 4.0.0)
##  farver                              2.0.3    2020-01-16 [1] CRAN (R 4.0.0)
##  fastmap                             1.0.1    2019-10-08 [1] CRAN (R 4.0.0)
##  generics                            0.0.2    2018-11-29 [1] CRAN (R 4.0.0)
##  GenomeInfoDb                      * 1.24.2   2020-06-15 [1] Bioconductor
##  GenomeInfoDbData                    1.2.3    2020-07-02 [1] Bioconductor
##  GenomicAlignments                   1.24.0   2020-04-27 [1] Bioconductor
##  GenomicFeatures                   * 1.40.0   2020-04-27 [1] Bioconductor
##  GenomicRanges                     * 1.40.0   2020-04-27 [1] Bioconductor
##  GenomicScores                     * 2.0.0    2020-04-27 [1] Bioconductor
##  ggplot2                           * 3.3.2    2020-06-19 [1] CRAN (R 4.0.0)
##  glue                                1.4.1    2020-05-13 [1] CRAN (R 4.0.0)
##  gtable                              0.3.0    2019-03-25 [1] CRAN (R 4.0.0)
##  gtools                              3.8.2    2020-03-31 [1] CRAN (R 4.0.2)
##  h2o                               * 3.32.0.3 2021-01-29 [1] local
##  HDF5Array                           1.16.1   2020-06-16 [1] Bioconductor
##  hms                                 0.5.3    2020-01-08 [1] CRAN (R 4.0.0)
##  htmltools                           0.5.0    2020-06-16 [1] CRAN (R 4.0.0)
##  httpuv                              1.5.4    2020-06-06 [1] CRAN (R 4.0.0)
##  httr                                1.4.1    2019-08-05 [1] CRAN (R 4.0.0)
##  interactiveDisplayBase              1.26.3   2020-06-02 [1] Bioconductor
##  IRanges                           * 2.22.2   2020-05-21 [1] Bioconductor
##  jsonlite                            1.7.0    2020-06-25 [1] CRAN (R 4.0.0)
##  knitr                               1.29     2020-06-23 [1] CRAN (R 4.0.0)
##  labeling                            0.3      2014-08-23 [1] CRAN (R 4.0.0)
##  later                               1.1.0.1  2020-06-05 [1] CRAN (R 4.0.0)
##  lattice                             0.20-41  2020-04-02 [1] CRAN (R 4.0.2)
##  lazyeval                            0.2.2    2019-03-15 [1] CRAN (R 4.0.0)
##  lifecycle                           0.2.0    2020-03-06 [1] CRAN (R 4.0.0)
##  magrittr                            1.5      2014-11-22 [1] CRAN (R 4.0.0)
##  Matrix                              1.2-18   2019-11-27 [1] CRAN (R 4.0.2)
##  matrixStats                         0.56.0   2020-03-13 [1] CRAN (R 4.0.0)
##  memoise                             1.1.0    2017-04-21 [1] CRAN (R 4.0.0)
##  mgcv                                1.8-31   2019-11-09 [1] CRAN (R 4.0.2)
##  mime                                0.9      2020-02-04 [1] CRAN (R 4.0.0)
##  munsell                             0.5.0    2018-06-12 [1] CRAN (R 4.0.0)
##  nlme                                3.1-148  2020-05-24 [1] CRAN (R 4.0.2)
##  openssl                             1.4.1    2019-07-18 [1] CRAN (R 4.0.0)
##  phastCons100way.UCSC.hg19         * 3.7.2    2020-07-02 [1] Bioconductor
##  pillar                              1.4.4    2020-05-05 [1] CRAN (R 4.0.0)
##  pkgconfig                           2.0.3    2019-09-22 [1] CRAN (R 4.0.0)
##  prettyunits                         1.1.1    2020-01-24 [1] CRAN (R 4.0.0)
##  progress                            1.2.2    2019-05-16 [1] CRAN (R 4.0.0)
##  promises                            1.1.1    2020-06-09 [1] CRAN (R 4.0.0)
##  ProtGenerics                        1.20.0   2020-04-27 [1] Bioconductor
##  purrr                               0.3.4    2020-04-17 [1] CRAN (R 4.0.0)
##  R6                                  2.4.1    2019-11-12 [1] CRAN (R 4.0.0)
##  rappdirs                            0.3.1    2016-03-28 [1] CRAN (R 4.0.0)
##  Rcpp                                1.0.4.6  2020-04-09 [1] CRAN (R 4.0.0)
##  RCurl                               1.98-1.2 2020-04-18 [1] CRAN (R 4.0.0)
##  rhdf5                               2.32.1   2020-06-18 [1] Bioconductor
##  Rhdf5lib                            1.10.0   2020-04-27 [1] Bioconductor
##  rlang                               0.4.6    2020-05-02 [1] CRAN (R 4.0.0)
##  rmarkdown                           2.3      2020-06-18 [1] CRAN (R 4.0.0)
##  Rsamtools                           2.4.0    2020-04-27 [1] Bioconductor
##  RSQLite                             2.2.0    2020-01-07 [1] CRAN (R 4.0.0)
##  rtracklayer                       * 1.48.0   2020-04-27 [1] Bioconductor
##  S4Vectors                         * 0.26.1   2020-05-16 [1] Bioconductor
##  scales                              1.1.1    2020-05-11 [1] CRAN (R 4.0.0)
##  sessioninfo                         1.1.1    2018-11-05 [1] CRAN (R 4.0.0)
##  shiny                               1.5.0    2020-06-23 [1] CRAN (R 4.0.0)
##  stringi                             1.4.6    2020-02-17 [1] CRAN (R 4.0.0)
##  stringr                             1.4.0    2019-02-10 [1] CRAN (R 4.0.0)
##  SummarizedExperiment                1.18.1   2020-04-30 [1] Bioconductor
##  tibble                              3.0.1    2020-04-20 [1] CRAN (R 4.0.0)
##  tidyselect                          1.1.0    2020-05-11 [1] CRAN (R 4.0.0)
##  TxDb.Hsapiens.UCSC.hg19.knownGene * 3.2.2    2020-07-02 [1] Bioconductor
##  vctrs                               0.3.1    2020-06-05 [1] CRAN (R 4.0.0)
##  WhistleR                          * 0.9.9    2021-01-30 [1] Bioconductor
##  withr                               2.2.0    2020-04-20 [1] CRAN (R 4.0.0)
##  xfun                                0.20     2021-01-06 [1] CRAN (R 4.0.2)
##  XML                                 3.99-0.3 2020-01-20 [1] CRAN (R 4.0.0)
##  xtable                              1.8-4    2019-04-21 [1] CRAN (R 4.0.0)
##  XVector                           * 0.28.0   2020-04-27 [1] Bioconductor
##  yaml                                2.2.1    2020-02-01 [1] CRAN (R 4.0.0)
##  zlibbioc                            1.34.0   2020-04-27 [1] Bioconductor
##
## [1] /Library/Frameworks/R.framework/Versions/4.0/Resources/library
```

# 5 References

1. Chen, Wei, et al. “iRNA-Methyl: Identifying N6-methyladenosine sites using pseudo nucleotide composition.” Analytical biochemistry 490 (2015): 26-33.
2. Cui, Xiaodong, et al. “Guitar: an R/Bioconductor package for gene annotation guided transcriptomic analysis of RNA-related genomic features.” BioMed research international 2016 (2016).
3. Wang, Yue, et al. “MetaTX: deciphering the distribution of mRNA-related features in the presence of isoform ambiguity, with applications in epitranscriptome analysis.” Bioinformatics (2020).
4. Linder, Bastian, et al. “Single-nucleotide-resolution mapping of m6A and m6Am throughout the transcriptome.” Nature methods 12.8 (2015): 767-772.
5. Chen, Kunqi, et al. “WHISTLE: a high-accuracy map of the human N 6-methyladenosine (m6A) epitranscriptome predicted using a machine learning approach.” Nucleic acids research 47.7 (2019): e41-e41.
6. Dominissini, Dan, et al. “Topology of the human and mouse m 6 A RNA methylomes revealed by m 6 A-seq.” Nature 485.7397 (2012): 201-206.
7. Li, X., Quon, G., Lipshitz, H. D., & Morris, Q. (2010). Predicting in vivo binding sites of RNA-binding proteins using mRNA secondary structure. Rna, 16(6), 1096-1107.
8. MacPherson, Q., Beltran, B., & Spakowitz, A. J. (2018). Bottom–up modeling of chromatin segregation due to epigenetic modifications. Proceedings of the National Academy of Sciences, 115(50), 12739-12744.