Wimtrap

Rivière Quentin qri@hotmail.be

Matthieu Defrance matthieu.dc.defrance@ulb.be

Massimiliano Corso, Madalina Ciortan, Nathalie Verbruggen

1. Description of the package

Wimtrap is a R package that allows to predict the location of transcription factor binding sites (TFBS) in a 'condition' -specific' manner. It aims at addressing a problematic of major importance in biology: the identification of genes that are regulated by a given transcription factor (TF). A popular approach consists of performing 'pattern-matching' analyses, which make use of the the sequence-specific affinities (motifs) of the transcription factors. However, relying solely on the detection of motif occurrences to predict binding sites lead to high false discovery rates in organisms such as plants and metazoans. Wimtrap implements a methodological workflow that exploits ChIP-chip/seq data in order to train models to predict the binding sites among motif occurrences detected by pattern-matching, in the studied condition (growth stage, tissue, cell type, treatment). These models integrate various genomic features at location of potential binding sites, such as the location on genes, the phylogenetic conservation of DNA sequence, the results of digital genomic footprinting or chromatin state features.

2. Installation of Wimtrap

Installing Wimtrap requires at first the installation of BiocManager, if it has not been done yet:

```
if(!require("BiocManager", quietly = TRUE)){
  install.packages("BiocManager")
}
```

The package can then be installed by typing the following:

```
BiocManager::install("RiviereQuentin/Wimtrap",
                     dependencies = TRUE,
                     build_vignettes = TRUE)
```

If an error occurs, it might be fixed in some cases by installing beforehand the dependence e1071:

```
if(!require("e1071", quietly = TRUE)){
  install.packages("e1071")
}
BiocManager::install("RiviereQuentin/Wimtrap",
                     dependencies = TRUE,
                     build_vignettes = TRUE)
```

3. Presentation of the package

3.1 Functions

Wimtrap makes available **4 functions** in order to **build** a **predictive model** of transcription factor binding sites (**TFBS**) from data obtained in the training organism-condition and to **obtain predictions** of the **location** of binding sites of the studied transcription factors in the studied organism-condition. These functions allow to perform the following steps in the workflow of the method:

Steps	Function
- **Import** into the R session of the **predictive genomic data** specific to the training and studied organism(s)-condition(s) - **Gathering** of the **transcript models** - **Location** of the **potential binding sites** of the training and studied transcription factor(s) along the genome	`importGenomicData()`
- **Genomic feature extraction** at the location of the potential binding sites, performed specifically to the training (studied) organism-condition for the training (studied) transcription factor(s) - **Feature scaling** and **categorical** variables **encoding**	`getTFBSdata()`
From the dataset related to the training transcription factor(s): - **Labeling** of the potential binding sites based on ChIP-seq/chip data obtained in the training organism-condition - **Dataset balancing** - **Machine learning** (XG Boosting) - **Model evaluation**	`buildTFBSmodel()`
From the dataset related to the studied transcription factor(s): - **Prediction** of the **location** of the **binding sites** and gene targets of the studied transcription factor(s) in the studied organism-condition	`predictTFBS()`

In addition, the package includes the `carepat()` function that allows to quickly get predictions of location of transcription factor binding sites in *Arabidopsis thaliana* and *Solanum lycopersicum*, in various conditions. This function makes use of pre-build general models obtained from seedling and flowers of Arabidopsis and ripening fruits of tomato.

3.2 Inputs and outputs

The use of Wimtrap can be further explained through the following diagram, which presents the main inputs to the method (in coral color) and the outputs of the `getTFBSdata()`, `buildTFBSmodel()` and `predictTFBS()` functions. Each input/output is presented with conceptual attributes and how they can be used by functions implemented by the package. The arrows express relationships of dependence.

```
ReferenceError: Can't find variable: Float64Array
```

This diagram illustrates how to practically address a problematic using Wimtrap:

1. Define the transcription factor(s) to study in a given organism and condition ('Condition' can be defined here by a growth stage, an organ, a treatment or any other criterion that allows to designate biological materials with comparable state of the chromatin).
2. Search for predictive genomic data specific to the organism and condition of study in the literature or on dedicated databases such as ArrayExpress or GEO. These genome-wide data can be related to results of phylogenetic footprinting (identification of conserved elements), digital genomic footrprinting and the chromatin state (degree of opening, histone modifications and variants, methylation of the cytosine).
    - *These data have to be downloaded in the BED or GTF/GFF format. The score field might be empty. The name field can be used to assign a category to the genomic intervals.*
    - *Transcript models can be automatically downloaded using the* `importGenomicData()` *function from the biomart database.*
3. Search for ChIP-chip/seq data related to transcription factor(s) and obtained in the studied organism-condition.
    - *The ChIP-chip/seq data have to be downloaded in the BED or NARROWPEAK format. N.B.: the score field might be empty as the score of the ChIP-peaks will not be considered.*
4. Are ChIP-chip/seq data available in the studied organism-condition?

- **YES**: the training organism-condition corresponds to the training organism-condition (optimal use case). Select all the available ChIP-chip/seq data for this organism-condition and go to the point 5.
- **NO**: define a training condition-organism different but close from the studied organism-condition. For the training condition-organism, gather predictive genomic data (a part or all of the genomic data gathered for the studided condition in point 2) and ChIP-seq/chip data related to the transcription factor(s) (if possible, select only thise related to the studied transcription factors so that TF-specific models can be built). Go to point the 5.
    - *The 'granularity' of the predictive genomic data has to be the same in the training and studied organism-condition. Scores have to be provided in both organisms-conditions or none. For categorical data, the category of genomic intervals have to be defined similarly in both organisms-conditions.*

5. Obtain the position frequency or weigth matrices (PFMs or PWMs) representing the motifs of the studied transcription factors and the studied ones (= those for which ChIP-chip/seq data will be considered to train the model), from literature search or dedicated databases such as Homer, Jaspar, Cis-BP, Transfac or PlantTFDB.

- *The PWMs/PFMs can be downloaded in the homer, cis-bp, jaspar, transfac, meme or raw PFM format.*
- *This step is not necessary if you want to locate transcription factor potential binding sites using other approaches (c.f* `matches` *argument of the* `getTFBSdata()` *function).*

6. Obtain the genome sequence of the studied and training organisms.

    - *The genome sequence(s) have to be downloaded in the FASTA format.*
    - *The genome sequences can be automatically downloaded using the* `getTFBSdata()` *function from the Ensembl or EnsemblGenomes database.*

7. Get the dataset of potential binding sites of the **studied** transcription factors: use the `importGenomicData()` to import into the session the predictive genomic data specific to the **studied** organism-condition and `getTFBSdata()` to locate the potential binding sites of the **studied** transcription factor(s) and annotate them with features extracted from the imported genomic data.

8. Get the dataset of potential binding sites of the **studied** transcription factors: use the `importGenomicData()` to import into the session the predictive genomic data specific to the **training** organism-condition and `getTFBSdata()` to locate the potential binding sites of the **training** transcription factor(s) and annotate them with features extracted from the imported genomic data.

    - *The pattern-matching can be performed using an external tool (c.f* `matches` *argument of the* `getTFBSdata()` *function) or can be achieved with Wimtrap using different p-value thresholds (c.f* `matches` *argument of the* `getTFBSdata()` *function).*
    - *The genomic features are extracted on windows, centered on the potential binding sites, of three different lenghts that might be set. Default: 20, 400 and 1000bp. (c.f* `small_window, medium_window, long_window` *arguments of the* `getTFBSdata()` *function)*

9. Train by extreme gradient boosting a predictive model based on the ChIP-chip/seq data and the dataset of the potential binding sites of the training transcription factors using the `buildTFBSmodel()` function.

    - *If ChIP-chip/seq data are available in the training organism-condition for the studied transcription factor(s), build a TF-specific model for each studied transcription factor. A TF-specific model can be obtained when the* `buildTFBSmodel()` *is fed with data related to only on transcription factor.*
    - *For advanced users, other machine learning methods can be implemented based on the ChIP-chip/seq data and the dataset of the potential binding sites of the training transcription factors using a custom code. The resulting object needs to be of a class that can be used by the function* `stats::predict()` *.*
    - *The length of the ChIP-peaks might be adjusted. Default = 400 bp (c.f* `lengthChIPpeaks` *argument of the* `buildTFBSmodel()` *function).*

10. Apply the (general or TF-specific) predictive model on the dataset of the potential binding of the studied transcription factor(s) to get the predictions specific to the organism-condition of interest.

    - *Each potential binding site is associated to a score (from 0 to 1) which can be seen as its likelihood to be a transcription factor binding site. By default, we recommand to predict as binding sites all the potential binding sites that get a score higher to 0.5.*

11. Use the predictions (1) to infer the potential gene targets of the studied transcription factor, (2) to assess the over-representation of potential binding sites among the promoters of co-regulated genes, (3) to compare the regulatory regions of different genes

4. Example: prediction of the binding sites of CCA1 in the roots of *Arabidopsis thaliana*

4.1. Use the `carepat()` function to apply a pre-built general model

Predictions for CCA1 binding sites in roots of *Arabidopsis thaliana* can be quickly obtained using a pre-built model obtained from seedlings of Arabidopsis. It is a general model, trained from data related to an extensive set of transcription factors.

```
CCA1predictions.roots <- carepat(organism = "Arabidopsis thaliana",
                                 condition = "roots",
                                 TFnames = "AT2G46830")#Use the AGI code of CCA1 to retrieve automatically its motif
from PlantTFDB
```

Predictions in other conditions (non-hair part of roots, flowers, seed coats, heat-shocked seedlings, seedlings exposed to various light treatments) can be obtained in Arabidopsis using `carepat()`. It is also possible to study the immature fruits and ripening fruits in *Solanum lycopersicum*.

In the section 4.2., we will illustrate the full workflow of Wimtrap from scratch and will build a TF-specific model obtained based on data related to

CCA1 only.

The output of `carepat()` is a `data.table` identic to that generated by `PredictTFBS()`. In the section 4.3., we provide some examples to manipulate such an output.

4.2. Build and apply a CCA1-specific model

4.2.1 Get the pre-requisites

Search for predictive data related to the studied organism-condition (roots of Arabidopsis)

- For 7-days old roots of *Arabidopsis thaliana*, we could find the results of DNAseI-seq and digital genomic footrprinting on the PlantTFDB database (Sullivan et al., 2014; Jin et al., 2017). These data will allow to extract genomic features of high predictivity of transcription factor binding sites. The results of DNAseI-seq consist of the location of regions of open chromatin, which are scored according to their degree of opening. Digital genomic footprinting is a method that is used to predict binding events on the DNA based on the protective effect for open DNA of bound proteins against DNAseI cleavage.
- We could also find condition-independent data related to the *Arabidopsis thaliana* species:
    - the location of conserved non-coding elements along the genome, from 3 different sources (Baxter et al., 2012; Haudry et al., 2013; Thomas et al., 2007).
    - the transcript models of *Arabidopsis thaliana*, which we will download from biomart (Kinsella et al., 2011) using the `importGenomicData()` function.

Search for training data

- We did not find any ChIP-seq/chip data obtained from roots of *Arabidopsis thaliana*: we will have to train a model from a training condition different from the studied one.
- We propose to build a model based on data related to seedlings of *Arabidopsis thaliana*:
    - ChIP-seq data about CCA1 in seedlings were published (Nagel et al., 2015): this will allow us to build a TF-specific model.
    - As for roots, we could find the results of DNAseI-seq and digital genomic footrprinting on the PlantTFDB database (Zhang et al., 2012; Sullivan et al., 2014; Jin et al., 2017).

Get the Pseudo Weight Matrix (PWM) of the motif of CCA1

CCA1 can bind the evening element, for which a PWM is made available on PlantTFDB (Jin et al., 2017).

Download the collected data

You can download all the data introduced above from the github repository RiviereQuentin/Wimtrap.

```
if(!require(utils)){
  install.packages(pkgs = "utils")
}
library(utils)
download.file(url = "https://github.com/RiviereQuentin/Wimtrap/raw/main/example.zip",
              destfile = "Wimtrap_ex.zip")
unzip(zipfile = "Wimtrap_ex.zip")
```

4.2.2. Apply the functions defined in the package

We are now ready to open our R session and apply the Wimtrap workflox to get predictions of the binding sites of CCA1 along the genome in 7-days old roots of *Arabidopsis thaliana*.

1. `importGenomicData()` to import the genomic data, for the seedlings and the roots of Arabidopsis:

```
library(Wimtrap)

#The file paths to the genomic data, encoded in BED or GTF/GFF files, are input through the `genomic_data` argument.
#Each file is named according to the feature that it allows to define.
#Remark: the chromosomes are named, in the chrom field of the BED files, according to their number.
#This number might be preceded by the prefix 'chr' (case-insensitive). For chromosome 1,  'chr1', 'CHR1', 'Chr1'
#or '1' are accepted.
#Remark: the regions described by a file are all assigned to a score of '1' if the score field is empty (cf. CNS).
# As for the genomic intervals that are not included in a file, they are all assigned to a null score.

imported_genomic_data.roots <- importGenomicData(organism = "Arabidopsis thaliana",
                                                 genomic_data = c(
                                                   DHS = "example/DHS_athal_roots_7_days.bed",
                                                   DGF = "example/DGF_athal_roots_7_days.bed",
                                                   CNS = "example/CNS_athal.bed"
                                                 ))
imported_genomic_data.seedlings <- importGenomicData(organism = "Arabidopsis thaliana",
                                                     genomic_data = c(
                                                      DHS = "example/DHS_athal_seedlings_normal.bed",
                                                      DGF = "example/DGF_athal_seedlings_7_days.bed",
                                                      CNS = "example/CNS_athal.bed"
                                                 ))
```

2. `getTFBSdata()` to build the dataset of potential binding sites of CCA1 both for the 7-days old roots and the seedlings of *Arabidopsis thaliana*: locate the matches with the evening element along the genome (default p-value = 0.001), extract the average signal from the predictive genomic data at location of the matches with the evening elements on centered windows of 3 different lengths (default = 20bp, 400bp and 1000bp) and finally, scale the extracted features between 0 and 1. To allow better comparison of the genomic features across organisms and conditions, the scaling is applied after detection of the outliers (\(>median+1.5*Interquartile range\) or \(<median-1.5*Interquartile range\)), whose values are replaced by \(median+1.5*Interquartile range\) or \(median-1.5*Interquartile range\).

The function will print a summary of the datasets of potential binding sites.

```
#The motif representing the evening element is encoded in a file in raw pfm format. Other formats are allowed:
#meme, jaspar, transfac, homer and cis-bp.

#You must specify the name of the transcription factor (here CCA1) as it appears in the file giving
#the motif throught the `TFnames` argument.

#The genome sequence of the considered organism(s) might be automatically downloaded if you provide
#their names through the `organism` argument.
#If you provide the genome sequence from a FASTA file, make sure that the chromosomes are named according to their
number.
#This number might be preceded by the prefix 'chr' (case-insensitive). For chromosome 1,  'chr1', 'CHR1', 'Chr1'
#or '1' are accepted.

CCA1data.roots <- getTFBSdata(pfm = "example/PFMs_athal.pfm",
                              TFnames = "CCA1",
                              organism = "Arabidopsis thaliana",
                              imported_genomic_data = imported_genomic_data.roots)
#> [1] "=> CCA1"
#>  seqnames      start               end              width   strand       matchScore       matchLogPval    Proximal
Promoter
#>  1:52657   Min.   :      88   Min.   :      95   Min.   :8   +:117422   Min.   :0.00000   Min.   :-4.211   Min.
:0.00000
#>  2:34815   1st Qu.: 6333525   1st Qu.: 6333532   1st Qu.:8   -: 87485   1st Qu.:0.00533   1st Qu.:-3.664   1st Qu.
:0.00000
#>  3:39408   Median :11882110   Median :11882117   Median :8   *:     0   Median :0.14035   Median :-3.407   Median
:0.00000
#>  4:31394   Mean   :12408940   Mean   :12408947   Mean   :8              Mean   :0.29777   Mean   :-3.467   Mean
```

```
:0.08899
#>  5:46633   3rd Qu.:17848530   3rd Qu.:17848536   3rd Qu.:8              3rd Qu.:0.57843   3rd Qu.:-3.212   3rd Qu.
:0.00000
#>            Max.   :30426658   Max.   :30426665   Max.   :8              Max.   :1.00000   Max.   :-3.104   Max.   :
1.00000
#>

#>     Promoter          X5UTR              CDS             Intron            X3UTR           Downstream           Clos
estTTS
#>  Min.   :0.0000   Min.   :0.00000   Min.   :0.0000   Min.   :0.00000   Min.   :0.00000   Min.   :0.0000   AT3G3318
7.1:   199
#>  1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.0000   AT3G3298
0.1:   196
#>  Median :0.0000   Median :0.00000   Median :0.0000   Median :0.00000   Median :0.00000   Median :0.0000   AT1G4012
9.1:   195
#>  Mean   :0.3938   Mean   :0.02246   Mean   :0.0483   Mean   :0.04672   Mean   :0.01904   Mean   :0.1163   AT1G4039
0.1:   153
#>  3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.0000   AT5G3267
0.1:   144
#>  Max.   :1.0000   Max.   :1.00000   Max.   :1.0000   Max.   :1.00000   Max.   :1.00000   Max.   :1.0000   AT1G4010
4.1:   140
#>                                                                                                          (Other)
:203880
#>  DistToClosestTTS       ClosestTSS     DistToClosestTSS     DHS_20bp          DGF_20bp          CNS_20bp          DH
S_400bp
#>  Min.   :0.0000   AT3G33187.1:   198   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min
.   :0.00000
#>  1st Qu.:0.0000   AT1G40129.1:   197   1st Qu.:0.07426   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st
Qu.:0.00000
#>  Median :0.2614   AT3G32980.1:   194   Median :0.24387   Median :0.00000   Median :0.00000   Median :0.00000   Med
ian :0.00000
#>  Mean   :0.2990   AT1G40390.1:   155   Mean   :0.31781   Mean   :0.02378   Mean   :0.03319   Mean   :0.04222   Mea
n   :0.07334
#>  3rd Qu.:0.4656   AT5G32670.1:   142   3rd Qu.:0.48692   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd
Qu.:0.00000
#>  Max.   :1.0000   AT4G08097.1:   141   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max
.   :1.00000
#>                   (Other)    :203880

#>    DGF_400bp         CNS_400bp      Matches_400bp      DHS_1000bp        DGF_1000bp        CNS_1000bp      Matches_
1000bp
#>  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.
:0.0000
#>  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.
:0.2000
#>  Median :0.00000   Median :0.0000   Median :0.2759   Median :0.00000   Median :0.00000   Median :0.02434   Median
:0.4000
#>  Mean   :0.07124   Mean   :0.1304   Mean   :0.2616   Mean   :0.11625   Mean   :0.10530   Mean   :0.19369   Mean
:0.4267
#>  3rd Qu.:0.00000   3rd Qu.:0.1319   3rd Qu.:0.4828   3rd Qu.:0.08138   3rd Qu.:0.03797   3rd Qu.:0.29209   3rd Qu.
:0.6000
#>  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.
:1.0000
#>
CCA1data.seedlings <- getTFBSdata(pfm = "example/PFMs_athal.pfm",
                                  TFnames = "CCA1",
                                  organism = "Arabidopsis thaliana",
                                  imported_genomic_data = imported_genomic_data.seedlings)
#> [1] "=> CCA1"
```

```
#>  seqnames      start               end               width   strand      matchScore       matchLogPval    ProximalP
romoter
#>  1:47180   Min.   :      88   Min.   :      95   Min.   :8   +:95711   Min.   :0.00000   Min.   :-4.189   Min.   :
0.00000
#>  2:30979   1st Qu.: 6308303   1st Qu.: 6308310   1st Qu.:8   -:87501   1st Qu.:0.01629   1st Qu.:-3.647   1st Qu.:
0.00000
#>  3:35224   Median :11906262   Median :11906270   Median :8   *:    0   Median :0.22762   Median :-3.380   Median :
0.00000
#>  4:28089   Mean   :12430137   Mean   :12430144   Mean   :8             Mean   :0.31311   Mean   :-3.518   Mean   :
0.08759
#>  5:41740   3rd Qu.:17937356   3rd Qu.:17937364   3rd Qu.:8             3rd Qu.:0.53120   3rd Qu.:-3.226   3rd Qu.:
0.00000
#>            Max.   :30426658   Max.   :30426665   Max.   :8             Max.   :1.00000   Max.   :-3.191   Max.   :1
.00000
#>

#>     Promoter          X5UTR              CDS              Intron            X3UTR           Downstream           Clo
sestTTS
#>  Min.   :0.0000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.0000   AT1G401
29.1:   185
#>  1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.0000   AT3G331
87.1:   177
#>  Median :0.0000   Median :0.00000   Median :0.00000   Median :0.00000   Median :0.00000   Median :0.0000   AT3G329
80.1:   174
#>  Mean   :0.3854   Mean   :0.02351   Mean   :0.05206   Mean   :0.04573   Mean   :0.01922   Mean   :0.1173   AT1G403
90.1:   150
#>  3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.0000   AT5G326
70.1:   133
#>  Max.   :1.0000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.0000   AT5G345
81.1:   133
#>                                                                                                          (Other)
:182260
#>  DistToClosestTTS       ClosestTSS     DistToClosestTSS     DHS_20bp          DGF_20bp          CNS_20bp          DH
S_400bp
#>  Min.   :0.0000   AT1G40129.1:   187   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min
.   :0.00000
#>  1st Qu.:0.0000   AT3G33187.1:   176   1st Qu.:0.07661   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st
Qu.:0.00000
#>  Median :0.2574   AT3G32980.1:   172   Median :0.24837   Median :0.00000   Median :0.00000   Median :0.00000   Med
ian :0.00000
#>  Mean   :0.2975   AT1G40390.1:   152   Mean   :0.32221   Mean   :0.02091   Mean   :0.08158   Mean   :0.04327   Mea
n   :0.06931
#>  3rd Qu.:0.4627   AT5G34581.1:   134   3rd Qu.:0.49399   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd
Qu.:0.00000
#>  Max.   :1.0000   AT5G32670.1:   131   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max
.   :1.00000
#>                   (Other)    :182260

#>    DGF_400bp         CNS_400bp       Matches_400bp       DHS_1000bp        DGF_1000bp        CNS_1000bp       Matches_100
0bp
#>  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.
0000
#>  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.
2000
#>  Median :0.0000   Median :0.0000   Median :0.4000   Median :0.0000   Median :0.0000   Median :0.02434   Median :0.
4000
#>  Mean   :0.1297   Mean   :0.1328   Mean   :0.3078   Mean   :0.1232   Mean   :0.1824   Mean   :0.19476   Mean   :0.
3738
#>  3rd Qu.:0.1590   3rd Qu.:0.1404   3rd Qu.:0.4000   3rd Qu.:0.1144   3rd Qu.:0.3096   3rd Qu.:0.29615   3rd Qu.:0.
6000
```

```
#>  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.
0000
#>
```

3. `buildTFBSmodel` to get the predictive classifier and, optionally, evaluate the model.

As we consider only one transcription factor here (CCA1), we will only be able to perform an 'internal' validation: balancing of the dataset (we will randomly select as many potential binnding sites that do not overlap a ChIP-peak of CCA1 in Arabidopsis seedlings than there are potential binding sites that do overla a ChIP-peak), we will use 80% of the binding sites to build the model and 20% to assess the model. However, if you are considering a set of training transcripion factors, it is best to select a few for performing an 'external' validation (cf. `TF4Validation` argument). These 'validation' transcription factors will be excluded from the training of the model and, after balancing of their cognate datasets, .

For the purposes of evaluation, the function will:

- plot the ROC achieved by the model in comparison with that achieved by the pure pattern-matching (i.e. when considering solely the p-value of the match). The area under the ROC of the model is written along the Y-axis.

  The ROC allows to illustrate the evaluation of the performances of a predictive method according to the minimum score threshold that is used to predict a potential binding site as a binding site (cf. the 'prediction score' for the modeling approach and the 'p-value of the match' for the pattern-matching analysis). The performances are evaluated in terms of sensitivity (Y-axis) and 1-specificity (X-axis), where \(sensitity=TP/(TP+FN)\) and \(specificity=TN/(TN+FP)\), with TP: true positive = number of ChIP-validated potential binding that have been correctly predicted as a binding site; FN: false negative = number of ChIP-validated potential binding that have been incorrectly predicted as not being a binding site; TN: true negative = number of ChIP-invalidated potential binding sites that have been correctly predicted as not being a binding site; FP: false positive = number of ChIP-invalidated potential binding sites that have been incorrectly predicted as a binding site.

  The area under the ROC (AUC) is an appropriate metrics for comparing the performances of predictive methods based on balanced datasets. Higher the AUC, more performant is a classifier. The AUC is comprised between 0 and 1 (a random guess corresponding to an AUC of 0.5).

- plot the feature importance, in terms of gain. The predictive model, trained by extreme gradien boosting, corresponds to an ensembl of decision trees that collectively 'vote' to predict the binding sites: their prediction scores are averaged. The gain of a feature is calculated by summing, on each decision tree, the increase of accuracy that is obtained on each branch that use this feature to classify the potential binding sites into the two classes considered ('binding site' or 'not binding site'). The gain is expressed relatively to all the features integrated by the model so that the total of the feature gains equal to 1.

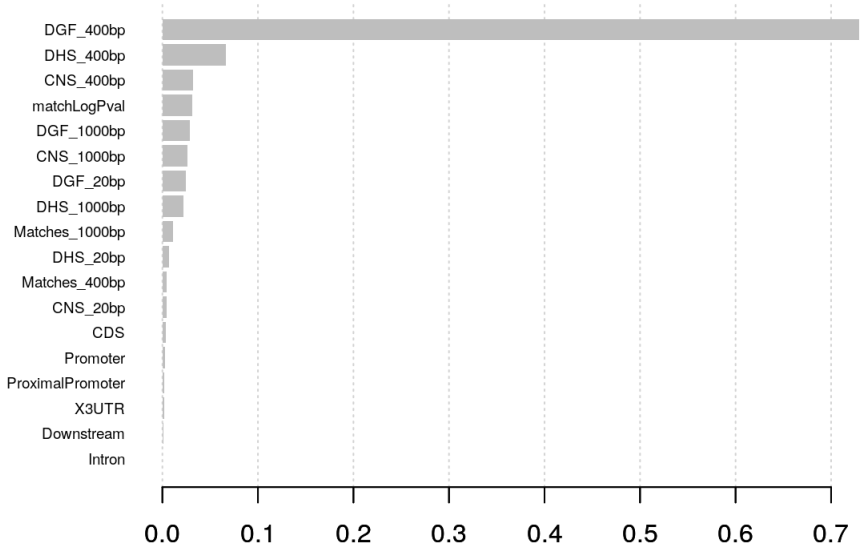
  The features are named as 'genomicdata_windowlength', i.e. from the genomic data that have been used to extract the feature (DHS, CNS,..) and the window length on which they have been extracted (by default 20bp, 400bp or 1000bp). For instance, we will obtain the feature 'DHS_20bp' for the feature obtained by averaging the signal of DNAseI-sensitity on the windows of 20bp centered on the potential binding sites. Noteworthy, the features related to the gene structures ('Promoter', 'CDS',..) are only named 'genomicdata' because we consider only the structure on whiche the center of the potential binding sites are located (for instance 'Promoter' = 1 if the center of the potential binding site is located on the promoter of the potential gene target; = 0 otherwise).

- print on the screen the confusion matrix obtained using a prediction score threshold of 0.5, that gives the FP, FN, TP and TN associated to the classification of the potential binding sites of the balanced 'validation' dataset by the model.

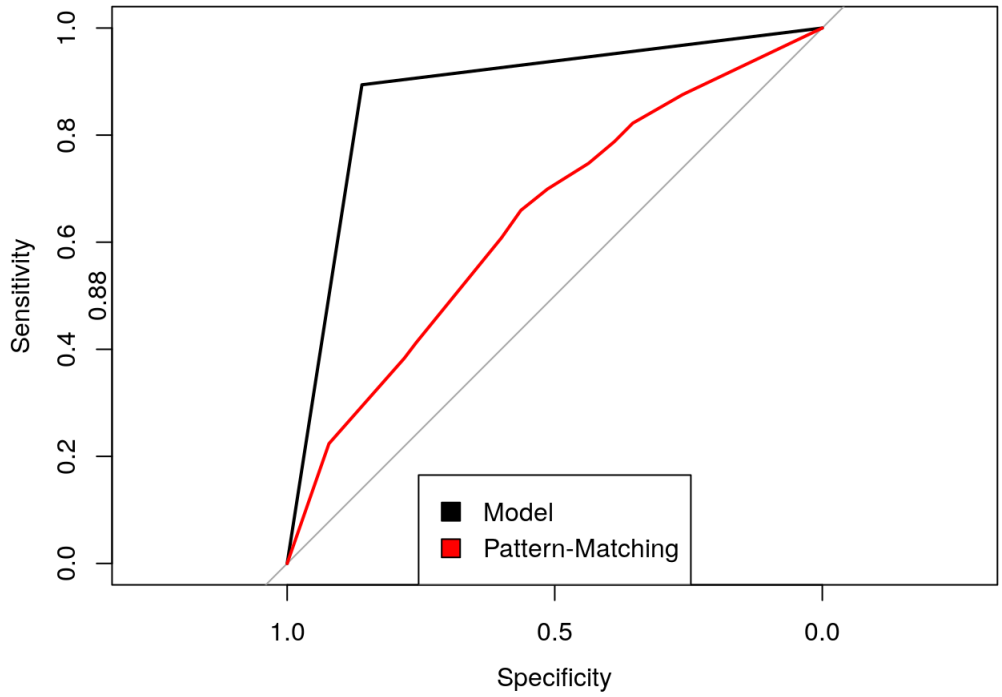
```
# Name the `ChIPpeaks` argument according to the training transcription factor(s)

CCA1model <- buildTFBSmodel(CCA1data.seedlings,
                            ChIPpeaks = c(CCA1 = "example/CCA1_athal_seedlings.narrowPeak"),
                            model_assessment = TRUE)
#> [1]  train-error:0.109347+0.001496   test-error:0.134174+0.004817
#> Multiple eval metrics are present. Will use test_error for early stopping.
#> Will train until test_error hasn't improved in 20 rounds.
#>
#> [11] train-error:0.089778+0.002343   test-error:0.125356+0.004523
#> [21] train-error:0.067427+0.001753   test-error:0.125356+0.002598
#> [31] train-error:0.051112+0.001105   test-error:0.127661+0.004555
#> [41] train-error:0.037749+0.000706   test-error:0.129018+0.004126
#> Stopping. Best iteration:
#> [23] train-error:0.064544+0.001674   test-error:0.124134+0.003921
#>
```

```
#> [23:59:00] WARNING: amalgamation/../src/learner.cc:516:
#> Parameters: { early_stop_round, print_every_n } might not be used.
#>
#>   This may not be accurate due to some parameters are only used in language bindings but
#>   passed down to XGBoost core.  Or some parameters are not used but slip through this
#>   verification. Please open an issue if you find above cases.
#>
#>
#> [1]  val-error:0.139989  train-error:0.109754
#> Multiple eval metrics are present. Will use train_error for early stopping.
#> Will train until train_error hasn't improved in 10 rounds.
#>
#> [11] val-error:0.125339  train-error:0.092660
#> [21] val-error:0.122626  train-error:0.071361
#> [23] val-error:0.122626  train-error:0.069326
#> Performances of the model
#> Confusion Matrix and Statistics
#>
#>           Reference
#> Prediction   0   1
#>          0 782  99
#>          1 127 835
#>
#>                Accuracy : 0.8774
#>                  95% CI : (0.8615, 0.892)
#>     No Information Rate : 0.5068
#>     P-Value [Acc > NIR] : < 2e-16
#>
#>                   Kappa : 0.7546
#>
#>  Mcnemar's Test P-Value : 0.07249
#>
#>             Sensitivity : 0.8603
#>             Specificity : 0.8940
#>          Pos Pred Value : 0.8876
#>          Neg Pred Value : 0.8680
#>              Prevalence : 0.4932
#>          Detection Rate : 0.4243
#>    Detection Prevalence : 0.4780
#>       Balanced Accuracy : 0.8771
#>
#>        'Positive' Class : 0
#>
#> Features importance             Feature         Gain       Cover    Frequency
#>  1:        DGF_400bp 0.7296545394 0.272879268 0.1461824953
#>  2:        DHS_400bp 0.0662213006 0.113760935 0.0828677840
#>  3:        CNS_400bp 0.0321700476 0.074176223 0.0977653631
#>  4:     matchLogPval 0.0315278137 0.116328729 0.0828677840
#>  5:       DGF_1000bp 0.0290440843 0.085742972 0.1098696462
#>  6:       CNS_1000bp 0.0259146524 0.050594416 0.1014897579
#>  7:         DGF_20bp 0.0245975837 0.059559884 0.0856610801
#>  8:       DHS_1000bp 0.0217980795 0.048365570 0.1014897579
#>  9:   Matches_1000bp 0.0115411087 0.015036058 0.0512104283
#> 10:         DHS_20bp 0.0065613470 0.032692689 0.0270018622
#> 11:    Matches_400bp 0.0048036903 0.007917049 0.0270018622
#> 12:         CNS_20bp 0.0045063455 0.029705580 0.0270018622
#> 13:              CDS 0.0034368035 0.032124664 0.0186219739
#> 14:         Promoter 0.0030661531 0.017026684 0.0139664804
#> 15: ProximalPromoter 0.0017786475 0.004499411 0.0074487896
#> 16:            X3UTR 0.0016817976 0.028938733 0.0111731844
```

```
#> 17:       Downstream 0.0014937469 0.007314459 0.0074487896
#> 18:           Intron 0.0002022588 0.003336675 0.0009310987
```




```
#>             Feature         Gain       Cover    Frequency   Importance
#>  1:        DGF_400bp 0.7296545394 0.272879268 0.1461824953 0.7296545394
#>  2:        DHS_400bp 0.0662213006 0.113760935 0.0828677840 0.0662213006
#>  3:        CNS_400bp 0.0321700476 0.074176223 0.0977653631 0.0321700476
#>  4:     matchLogPval 0.0315278137 0.116328729 0.0828677840 0.0315278137
#>  5:       DGF_1000bp 0.0290440843 0.085742972 0.1098696462 0.0290440843
#>  6:       CNS_1000bp 0.0259146524 0.050594416 0.1014897579 0.0259146524
#>  7:         DGF_20bp 0.0245975837 0.059559884 0.0856610801 0.0245975837
#>  8:       DHS_1000bp 0.0217980795 0.048365570 0.1014897579 0.0217980795
#>  9:   Matches_1000bp 0.0115411087 0.015036058 0.0512104283 0.0115411087
#> 10:         DHS_20bp 0.0065613470 0.032692689 0.0270018622 0.0065613470
#> 11:    Matches_400bp 0.0048036903 0.007917049 0.0270018622 0.0048036903
#> 12:         CNS_20bp 0.0045063455 0.029705580 0.0270018622 0.0045063455
#> 13:              CDS 0.0034368035 0.032124664 0.0186219739 0.0034368035
#> 14:         Promoter 0.0030661531 0.017026684 0.0139664804 0.0030661531
#> 15: ProximalPromoter 0.0017786475 0.004499411 0.0074487896 0.0017786475
#> 16:            X3UTR 0.0016817976 0.028938733 0.0111731844 0.0016817976
#> 17:       Downstream 0.0014937469 0.007314459 0.0074487896 0.0014937469
#> 18:           Intron 0.0002022588 0.003336675 0.0009310987 0.0002022588
```




```
#>
#> Call:
#> roc.default(response = ts_label, predictor = xgbpred)
#>
#> Data: xgbpred in 909 controls (ts_label 0) < 934 cases (ts_label 1).
#> Area under the curve: 0.8771
#>
#> Call:
#> roc.default(response = ts_label, predictor = test$matchLogPval)
#>
#> Data: test$matchLogPval in 909 controls (ts_label 0) > 934 cases (ts_label 1).
#> Area under the curve: 0.6425
#> $rect
#> $rect$w
#> [1] -0.5087035
#>
#> $rect$h
#> [1] 0.2050633
#>
#> $rect$left
#> [1] 0.7543517
#>
#> $rect$top
#> [1] 0.1650633
#>
#>
#> $text
#> $text$x
#> [1] 0.6337134 0.6337134
#>
#> $text$y
#> [1] 0.09670886 0.02835443
#>
#>
#> NULL
```

4. `predictTFBS()` to predict the targets of CCA1 in roots of Arabidopsis thaliana with a given prediction score threshold (default = 0.5).

```
CCA1.predictions.roots <- predictTFBS(CCA1model, CCA1data.roots)
head(CCA1.predictions.roots)
#>    seqnames start   end width strand  transcript prediction.score   TF
#> 1:        1  3081  3088     8      + AT1G01010.1        0.7871000 CCA1
#> 2:        1 15408 15415     8      + AT1G01030.2        0.8014211 CCA1
#> 3:        1 15499 15506     8      + AT1G01030.2        0.8301628 CCA1
#> 4:        1 38293 38300     8      + AT1G01060.5        0.6774676 CCA1
#> 5:        1 49340 49347     8      + AT1G01090.1        0.6646637 CCA1
#> 6:        1 59179 59186     8      + AT1G01120.1        0.9512815 CCA1
```

The output is a `data.table` indicating for each predicted binding site: * the coordinates (‘`seqnames`’ = chromosome name, ‘`start`’ and ‘`end`’= start and end on the chromosome), the width (‘`width`’) and the orientation of the predicted binding sites (‘`strand`’); * the name of the potential transcript targets, ‘`transcript`’ (the transcript whose the transcription start is the closest); * the prediction score (`prediction.score`) output by the model (comprised between the minimum score threshold and 1); * the name of the related transcription factor (`TF`).

Optionally, the `data.table` might include the annotations of the predicted binding sites with the predictive features. To obtain this information, call `predictTFBS()` while setting `show_annotations = TRUE`.

```
CCA1.annotations.predictions.roots <- predictTFBS(CCA1model, CCA1data.roots, show_annotations = TRUE)
head(CCA1.annotations.predictions.roots)
```

4.3. Manipulate the predictions

4.3.1. Filter the predictions

It is possible to further filter the predicted binding sites with simple manipulations of the `data.table` output by `predictTFBS()`, according for instance to the related transcription factor (here, we considered only CCA1 but we could have obtained predictions for multiple transcription factors), the potentiel target or the predictive features, the prediction score or the predictive features.

As a first example, we will verify if there are predicred binding sites on *AT3G46640*, which is a known target of CCA1.

```
OnAT3G46640 <- CCA1.predictions.roots[grep(pattern = "AT3G46640", CCA1.predictions.roots$transcript),]
print(OnAT3G46640)
#>    seqnames    start      end width strand  transcript prediction.score   TF
#> 1:        3 17182990 17182997     8      + AT3G46640.1        0.9811640 CCA1
#> 2:        3 17183484 17183491     8      + AT3G46640.2        0.5947698 CCA1
#> 3:        3 17182972 17182979     8      - AT3G46640.1        0.9914682 CCA1
```

We can observe that, as expected, two binding sites have been predicted, with very high prediction scores (almost 1, the maximum).

As a second example, we will select only the binding sites that overlap a conserved non-coding element (CNS). To do so, we will need to consider the `data.table` obtained with `predictTFBS()` with the option `show_annotations = TRUE`. We will put as condition that the feature `CNS_20bp` (here, the percentage of coverage by CNS of the windows of 20bp centered on the predicted binding sites) is superior to 0. Such an approach can be meaningful because the phylogenetic footprint is a good indicator of functionality (regulatory role) of the binding sites (Rister et al., 2010), as is the location on a proximal promoter (Li et al., 2019). The model does not take indeed into account whether the recruitment of the transcription on a predicted binding site is likely to impact or not the expression of the target gene.

```
OnCNS <- CCA1.annotations.predictions.roots[CCA1.annotations.predictions.roots$CNS_20bp > 0,]
head(OnCNS)
#>    seqnames  start    end width strand matchScore matchLogPval ProximalPromoter Promoter X5UTR CDS Intron X3UTR Do
wnstream  ClosestTTS
#> 1:        1   9806   9813     8      + 0.09623764    -3.267970               0        1     0   0      0     0
0 AT1G01030.1
#> 2:        1  15499  15506     8      + 0.00000000    -3.203460               0        1     0   0      0     0
0 AT1G01030.2
#> 3:        1  79725  79732     8      + 0.09111956    -3.236051               0        0     0   0      0     0
0 AT1G01180.1
#> 4:        1  99718  99725     8      + 0.09111956    -3.236051               1        1     0   0      0     0
0 AT1G01230.1
#> 5:        1 185091 185098     8      + 1.00000000    -4.185231               0        0     1   0      0     0
0 AT1G01500.1
#> 6:        1 201739 201746     8      + 0.27576754    -3.461408               1        1     0   0      0     0
0 AT1G01550.1
#>    DistToClosestTTS  transcript DistToClosestTSS DHS_20bp  DGF_20bp CNS_20bp  DHS_400bp DGF_400bp CNS_400bp Matche
s_400bp DHS_1000bp
#> 1:        0.6039303 AT1G01020.1        0.1389307  0.00000 0.0000000     0.30 0.07259040 0.2120800 0.8982456
0.0 0.09027778
#> 2:        0.0000000 AT1G01030.2        0.0000000  0.43100 0.8737807     0.35 0.44567611 0.5484212 1.0000000
0.0 0.29930556
#> 3:        0.7756126 AT1G01180.1        0.9758563  0.00000 0.0000000     0.05 0.18923139 0.2576477 0.9754386
0.4 0.25416667
#> 4:        0.3806117 AT1G01240.4        0.2284879  0.90345 1.0000000     0.35 0.98338280 1.0000000 0.7368421
0.0 1.00000000
#> 5:        0.0000000 AT1G01500.1        0.2612364  0.00000 0.4088546     0.80 0.79966635 0.7876883 0.5473684
0.0 0.69444444
#> 6:        0.2975662 AT1G01560.3        0.1922306  0.00000 0.0000000     0.85 0.02214249 0.1084009 1.0000000
0.4 0.10138889
#>    DGF_1000bp CNS_1000bp Matches_1000bp prediction.score   TF
#> 1: 0.18317423  1.0000000            0.4        0.6007259 CCA1
#> 2: 0.50088702  0.8681542            0.6        0.8405901 CCA1
#> 3: 0.24825024  1.0000000            0.8        0.8159643 CCA1
#> 4: 0.99530390  1.0000000            0.0        0.8792665 CCA1
#> 5: 0.81260488  0.6612576            0.0        0.9718286 CCA1
#> 6: 0.08018213  0.7261663            0.2        0.5928553 CCA1
```

Noteworthy, the `data.table` output by `predictTFBS()` can easily be turned into a `GRanges` object using the `GenomicRanges::makeGRangesFromDataFrame()` function. This allows the manipulation of the genomic intervals and, for instance, select binding sites based on genomic coordinates of interest.

Let's say that we want to select the binding sites located on chromosome 2, from 10000 to 20000bp and chromosome 3, from 30000 to 60000, on both strands:

```
library(GenomicRanges)

#Check before how are named the chromosomes in the predictions
levels(CCA1.predictions.roots$seqnames)
#> NULL

#Chromosome 2 is named '2' chromosome 3, '3'

RegionsOfInterest <- GRanges(seqnames = c(2,3),#names of the chromosomes of interest
                             ranges = c(IRanges(start = 10000, end = 20000),
                                        IRanges(start = 30000, end = 60000)),
                             strand = "*") #indicates that the strand does not matter

OnRegionsOfInterest <- subsetByOverlaps(makeGRangesFromDataFrame(CCA1.predictions.roots, keep.extra.columns = TRUE),
                                        RegionsOfInterest)
print(OnRegionsOfInterest)
#> GRanges object with 4 ranges and 3 metadata columns:
#>       seqnames      ranges strand |  transcript  prediction.score          TF
#>          <Rle>   <IRanges>  <Rle> |    <factor>         <numeric> <character>
#>   [1]        3 45828-45835      + | AT3G01130.7 0.825993955135345        CCA1
#>   [2]        3 59372-59379      + | AT3G01175.1 0.768293917179108        CCA1
#>   [3]        3 42246-42253      - | AT3G01120.1 0.553467392921448        CCA1
#>   [4]        3 51599-51606      - | AT3G01150.4 0.938434600830078        CCA1
#>   -------
#>   seqinfo: 5 sequences from an unspecified genome; no seqlengths
```

We can observe that there are three predicted binding sites on the 3:3000-60000 region and none on the 2:10000-20000.

4.3.2. Infer the potential targets of the studied transcription factors in the condition considered

A transcript is considered as a potential target if its transcription start site is the closest to a predicted binding site of a studied transcription factor in the condition considered. The whole set of potential transcript targets can be found in the column `transcript` of the `data.table` output by `predictTFBS()`, after selecting the binding sites related to the transcription factor of interest (which is necessary to do in the case where you obtained predictions for several transcription factors).

```
CCA1.potentialtargets <- CCA1.predictions.roots$transcript[CCA1.predictions.roots$TF=="CCA1"] #Be sure to select the transcription of interest
CCA1.potentialtargets <- CCA1.potentialtargets[!duplicated(CCA1.potentialtargets)] #remove the duplicated
head(CCA1.potentialtargets)
#> [1] AT1G01010.1 AT1G01030.2 AT1G01060.5 AT1G01090.1 AT1G01120.1 AT1G01180.1
#> 34840 Levels: AT1G01010.1 AT1G01020.1 AT1G01020.6 AT1G01030.1 AT1G01030.2 AT1G01040.1 AT1G01040.2 AT1G01050.1 AT1G01050.2 ... AT5G67640.1
```

If you want to simplify the predictions at the gene level, eliminate the ".[1..9]" part of the transcript names:

```
CCA1.potentialtargets <- unlist(strsplit(as.character(CCA1.potentialtargets), "[.]"))[seq(1,2*length(CCA1.potentialtargets),2)]
CCA1.potentialtargets <- CCA1.potentialtargets[!duplicated(CCA1.potentialtargets)]
head(CCA1.potentialtargets)
#> [1] "AT1G01010" "AT1G01030" "AT1G01060" "AT1G01090" "AT1G01120" "AT1G01180"
```

Whenever possible, we encourage you to further filter the predicted gene targets based on expression data that allow to assess the regulatory role of the studied transcription factor(s).

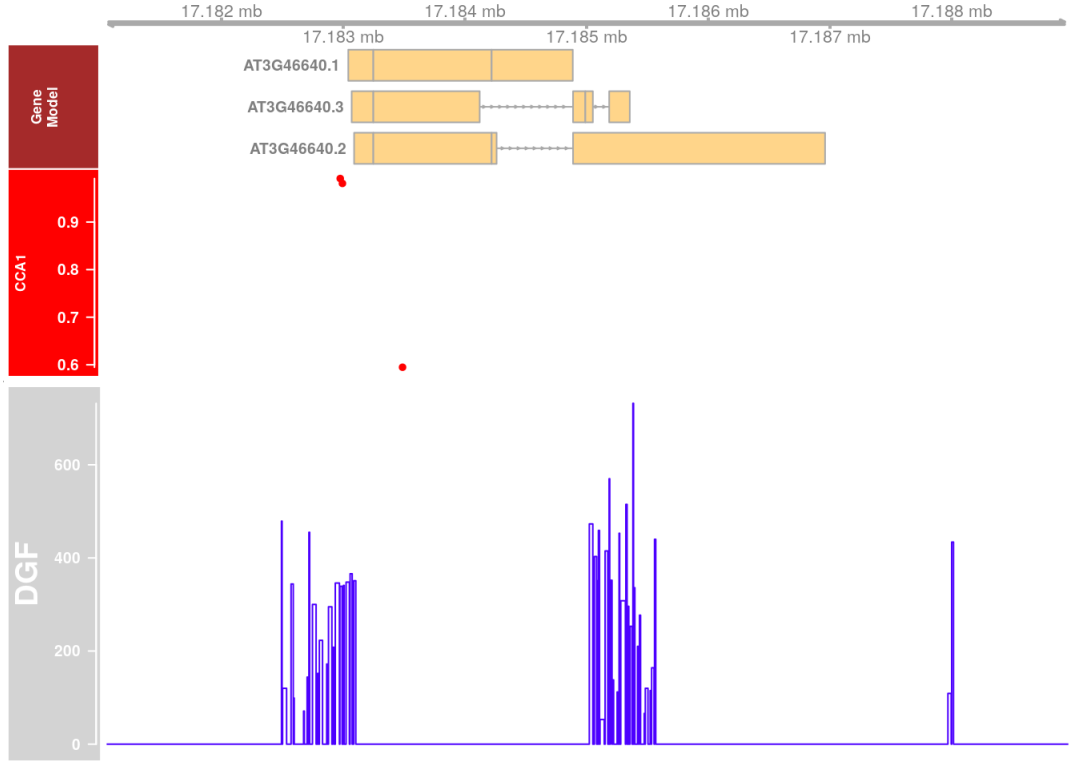
The prediction of the ene targets of a transcription factor can be used to get functional insights of the latter, by performing a Gene Ontolgy or Gene Sets (as sets of co-expressed or co-regulated genes) enrichment analyses.

4.3.3. Plot the location of the predicted binding sites on a gene of interest

We can vizualize, on a potential gene target of interest, the location and the prediction score of the predicted binding sites of the studied transcription

factors, in the condition considered (here, of CCA1 in roots). Additionally, you can plot the signals of genomic-wide data, such as the DGF, which is a feature of high predictivity (see below the plot of feature importances). Let's take as an example again the known target of CCA1 *AT3G46640*:

```
plotPredictions(CCA1.predictions.roots,
                imported_genomic_data = imported_genomic_data.roots,
                gene = "AT3G46640",
                genomic_data = "DGF")
```



Such plots can help to choose mutation sites or to compare graphically the potential cis-regulatory landscape of different genes.

5. Customization of the implementation

The package has been thought so that it allows changes in the default implementation.

5.1. Skip the downloading of the data related to gene structures and/or that of the genome sequence

The functions `importGenomicData()` and `getTFBSdata()` offer functionalities of downloading of gene structures data (location of the promoter, coding sequence,... of the transcript models) and genome sequence. This might bring a gain of time and the ensurance that the related data are correctly input into the workflow. However, access to internet is not always possible, the downloading can take several minutes while the inputting from source files is instantaneous and custom data might be necessary in some cases (for instance, you might need an old assembly of a genome).

An example of inputting the data entirely from source files is given by the help pages of the functions `importGenomicData()` (cf. arguments `genmic_data`, `tss` and `tts`) and `getTFBSdata()` (cf. argument `genome_sequence`).

We will predict the binding sites of PIF3 and TOC1 in Arabidopsis seedlings based on a toy genome (obtained by random subsetting of the genome of Arabidopsis).

The location of the 5'untranslated regions (5'UTR), coding sequences (CDS), introns and 3'UTR need to be imported from BED files through the `genomic_data` argument of `importGenomicData()`. It is important that the vector passed to `genomic_data` names the paths to these files with exactly the following names: "X5UTR", "CDS", "Intron" and "X3UTR". The location of transcription start site (TSS) and transcription termination site (TTS) are input throught the `tss` and `tts` arguments of the `importGenomicData()` from BED files. In the all the abovementioned files, the intervals are named in the 'name' field according to the name of the treanscript that they compose. It is not necessary to give the location of the promoters, proxomal promoters and downstream regions as this is calculated automatically based on the location of the TSS and TTS and the lengths of promoters, proximal promoters and downstream regions that are set throught the arguments `promoter_length` (default = 2000bp), `proximal_length` (default = 500bp) and `downstream_length` (default = 1000bp) of `importGenomicData()`.

The genome sequence can be input from a fasta file (that might be compressed) through the `genome_sequence` of the `getTFBSdata()` function.

```
#Pay attention to the names of the genomic_data.ex. Use exactly the names "X5UTR", "CDS", "Intron" and "X3UTR" for t
he paths to the files related to the 5'UTRs, CDS, introns and 3'UTRs.
#The system of nomenclature of the transcripts need to be coherent across the different files that are imported.
genomic_data.ex <- c(CE = system.file("extdata/conserved_elements_example.bed", package = "Wimtrap"),
                     DGF = system.file("extdata/DGF_example.bed", package = "Wimtrap"),
                     DHS = system.file("extdata/DHS_example.bed", package = "Wimtrap"),
                     X5UTR = system.file("extdata/x5utr_example.bed", package = "Wimtrap"),
                     CDS = system.file("extdata/cds_example.bed", package = "Wimtrap"),
                     Intron = system.file("extdata/intron_example.bed", package = "Wimtrap"),
                     X3UTR = system.file("extdata/x3utr_example.bed", package = "Wimtrap")
                    )
imported_genomic_data.ex <- importGenomicData(biomart = FALSE,
                                              genomic_data = genomic_data.ex,
                                              tss = system.file("extdata/tss_example.bed", package = "Wimtrap"),
                                              tts = system.file("extdata/tts_example.bed", package = "Wimtrap"))
TFBSdata.ex <- getTFBSdata(pfm = system.file("extdata/pfm_example.pfm", package = "Wimtrap"),
                           TFnames = c("PIF3", "TOC1"),
                           organism = NULL,
                           genome_sequence = system.file("extdata/genome_example.fa", package = "Wimtrap"),
                           imported_genomic_data = imported_genomic_data.ex)
```

5.2. Manipulate the genomic data and the datasets of potential binding sites

The objects output by `importGenomicData()` and `getTFBSdata()` can be manipulated to customize the predictive genomic features. The output of `importGenomicData()` is a list of GRanges objects that can be individually modified before feeding the `getTFBSdata()` function. The ouutput of `getTFBSdata()` is a vector of file paths that encode the dataset of binding sites for each considered transcription factor. The datasets can be imported in R, modified and re-exported in a tab-separated file before calling `buildTFBSmodel`

This might especially allow to define features that takes into account the differences of signal between two growth stages, two treatments, two organs... As an example, we will build a model specific to CCA1 that takes integrates features that calculate the differences between the average DGF scores extracted on windows of 20bp, 400bp and 1000bp centered on the potential binding sites in the roots and in the seedlings.

```
tmp.roots <- data.table::fread(CCA1data.roots)
tmp.seedlings <- data.table::fread(CCA1data.seedlings)

tmp.roots <- cbind(tmp.roots,
                       DGF_20bp_diff = tmp.seedlings$DGF_20bp-tmp.roots$DGF_20bp,
                       DGF_400bp_diff = tmp.seedlings$DGF_400bp-tmp.roots$DGF_400bp,
                       DGF_1000bp_diff = tmp.seedlings$DGF_1000bp-tmp.roots$DGF_1000bp)
tmp.seedlings <- cbind(tmp.seedlings,
                       DGF_20bp_diff = tmp.seedlings$DGF_20bp-tmp.roots$DGF_20bp,
                       DGF_400bp_diff = tmp.seedlings$DGF_400bp-tmp.roots$DGF_400bp,
                       DGF_1000bp_diff = tmp.seedlings$DGF_1000bp-tmp.roots$DGF_1000bp)

data.table::fwrite(tmp.roots,
                   "Diff_CCA1_roots.tsv",
                   sep = "\t")

data.table::fwrite(tmp.seedlings,
                   "Diff_CCA1_seedlings.tsv",
                   sep = "\t")


Diff.model <- buildTFBSmodel(c(CCA1 = "Diff_CCA1_seedlings.tsv"), #name the file path ("CCA1=...")
                             ChIPpeaks = c(CCA1 = "example/CCA1_athal_seedlings.narrowPeak"))

Diff.predictions <- predictTFBS(Diff.model, c(CCA1 = "Diff_CCA1_roots.tsv"))
```

5.3. Use a different algorithm of pattern-matching

It is possible to bypass the step of pattern-matching by indicating directly the location of potential binding sites as a `GRanges` object directly to `getTFBSdata()` through the `matches` argument. This allows to import, for instance, the results of pattern-matching obtained from tools external to Wimtrap.

Let's consider here a set of 1000 potential binding sites defined randomly on the chromosome 1 of Arabidipsis for whic we will assign a random score. The raw score of the potential binding need to be input in the first metadata column and the log10 of the p-value in the second.

```
library(GenomicRanges)
RandomPotentialTFBS <- GRanges(seqnames = 1,
                               ranges = IRanges(start = runif(1000, min = 1, max = 30427671),
                                                width = 12),
                               strand = "*")
mcols(RandomPotentialTFBS)[,"matchScore"] <- rnorm(1000, 0.5, 0.1)
mcols(RandomPotentialTFBS)[,"matchLogPval"] <- -abs(rnorm(1000, 4, 0.05) )
RandomData <- getTFBSdata(matches = c(CCA1 = tmp),
                          imported_genomic_data = imported_genomic_data.seedlings)
```

5.4. Use a different machine learning algorithm

You can use the `buildTFBSmodel()` function to carry out only the steps of labelling, balancing and splitting between a training and validation datasets by setting the argument `xgb_modeling` to `FALSE`. In that case, you will obtain a list of `data.tables` corresponding to the training and the validation datasets. The training dataset can be easily used with your favorite machine learning algorithm, the `ChIP-peak` column defining the target variable (=1 if the potential binding site is validated by a ChIP-peak; =0 otherwise). When you will obtain your model, you will be able to assess it with the validation dataset.

```
CCA1.training.validation.sets <- buildTFBSmodel(CCA1data.seedlings,
                                                ChIPpeaks = c(CCA1 = "example/CCA1_athal_seedlings.narrowPeak"),
                                                xgb_modeling = FALSE)
training.dataset <- CCA1.training.validation.sets$training.dataset
validation.dataset <- CCA1.training.validation.sets$validation.dataset
```

References

Baxter, L. et al. Conserved Noncoding Sequences Highlight Shared Components of Regulatory Networks in Dicotyledonous Plants. Plant Cell 24, 3949–3965 (2012).

Haudry, A. et al. An atlas of over 90,000 conserved noncoding sequences provides insight into crucifer regulatory regions. Nat. Genet. 45, 891–898 (2013).

Jin, J. et al. PlantTFDB 4.0: toward a central hub for transcription factors and regulatory interactions in plants. Nucleic Acids Res. 45, D1040–D1045 (2017).

Li, H., Quang, D. & Guan, Y. Anchor: trans-cell type prediction of transcription factor binding sites. Genome Res. 29, 281–292 (2019)

Kinsella, R. J. et al. Ensembl BioMarts: a hub for data retrieval across taxonomic space. Database 2011, bar030–bar030 (2011).

Nagel, D. H. et al. Genome-wide identification of CCA1 targets uncovers an expanded clock network in Arabidopsis. Proc. Natl. Acad. Sci. 112, E4802–E4810 (2015).

Rister, J. & Desplan, C. Deciphering the genome's regulatory code: The many languages of DNA. BioEssays 32, 381–384 (2010).

Thomas, B. C., Rapaka, L., Lyons, E., Pedersen, B. & Freeling, M. Arabidopsis intragenomic conserved noncoding sequence. Proc. Natl. Acad. Sci. 104, 3348–3353 (2007).

Sullivan, A. M., Arsovski, A.A., Lempe, J., Bubb, K.L. et al. Mapping and dynamics of regulatory DNA and transcription factor networks in A. thaliana. Cell Rep. 8, 2015-2030 (2014).

Zhang, W., Zhang, T., Wu, Y. & Jiang, J. Genome-Wide Identification of Regulatory DNA Elements and Protein-Binding Footprints Using Signatures of Open Chromatin in Arabidopsis. Plant Cell 24, 2719–2731 (2012).