Package 'LedPred'

April 23, 2016

Title Learning from DNA to Predict enhancers

Description This package aims at creating a predictive model of regulatory sequences used to score unknown sequences based on the content of DNA motifs, next-generation sequencing (NGS) peaks and signals and other numerical scores of the sequences using supervised classification. The package contains a workflow based on the support vector machine (SVM) algorithm that maps features to sequences, optimize SVM parameters and feature number and creates a model that can be stored and used to score the regulatory potential of unknown sequences.

Version 1.2.1

Date 2016-03-26

Author Elodie Darbo, Denis Seyres, Aitor Gonzalez

Maintainer Aitor Gonzalez <aitor.gonzalez@univ-amu.fr>

Depends R (>= 3.2.0), e1071 (>= 1.6)

Imports akima, GenomicRanges (>= 1.18.4), irr, jsonlite, parallel, plot3D, plyr, RCurl, ROCR, testthat

License MIT | file LICENSE

LazyData true

biocViews SupportVectorMachine, Software, MotifAnnotation, ChIPSeq

NeedsCompilation no

BugReports https://github.com/aitgon/LedPred/issues

RoxygenNote 5.0.0

R topics documented:

`createModel` *Create the model with the optimal features*

Description

`createModel` function creates a SVM model from the training data set with the selected features.

Usage

```
createModel(data = NULL, data.granges = NULL, cl = 1, kernel = "radial",
  scale = FALSE, cost = 1, gamma = 1, valid.times = 10,
  feature.ranking = NULL, feature.nb = NULL, file.prefix = NULL)
```

Arguments

`data`	data.frame containing the training set
`data.granges`	Bioconductor GenomicRanges object containing the training set
`cl`	integer indicating the column number corresponding to the response vector that classify positive and negative regions (default = 1)
`kernel`	SVM kernel, a character string: "linear" or "radial". (default = "radial")
`scale`	Logical indicating if the data have to be scaled or not (default = FALSE)
`cost`	The SVM cost parameter for both linear and radial kernels. If NULL (default), the function `mcTune` is run.
`gamma`	The SVM gamma parameter for radial kernel. If radial kernel and NULL (default), the function `mcTune` is run.
`valid.times`	Integer indicating how many times the training set will be split for the cross validation step (default = 10). This number must be smaller than positive and negative sets sizes.
`feature.ranking`	List of ordered features.
`feature.nb`	the optimal number of feature to use from the list of ordered features.
`file.prefix`	A character string that will be used as a prefix followed by "_model.RData" for the resulting model file, if it is NULL (default), no model is saved

Value

the best SVM model

Examples

```
    data(crm.features)
    cost <- 1
    gamma <- 1
    data(feature.ranking)
    feature.nb <- 70
#svm.model <- createModel(data.granges=crm.features, cost=cost, gamma=gamma,
#    feature.ranking=feature.ranking, feature.nb=feature.nb)
#feature.weights <- as.data.frame(t(t(svm.model$coefs) %*% svm.model$SV))
```

`crm.features` *This is data to be included in my package*

Description

This is data to be included in my package

`evaluateModelPerformance`
Evaluate model performances

Description

`evaluateModelPerformance` function computes the precision and recall measures to evaluate the model through cross validation steps using `ROCR` package.

Usage

```
evaluateModelPerformance(data = NULL, data.granges = NULL, cl = 1,
  valid.times = 10, feature.ranking = NULL, feature.nb = NULL,
  numcores = parallel::detectCores() - 1, file.prefix = NULL,
  kernel = "linear", scale = FALSE, cost = 1, gamma = 1)
```

Arguments

`data`	data.frame containing the training set
`data.granges`	Bioconductor GenomicRanges object containing the training set
`cl`	integer indicating the column number corresponding to the response vector that classify positive and negative regions (default = 1)
`valid.times`	Integer indicating how many times the training set will be split for the cross validation step (default = 10). This number must be smaller than positive and negative sets sizes.
`feature.ranking`	List of ordered features.

feature.nb	the optimal number of feature to use from the list of ordered features.
numcores	Number of cores to use for parallel computing (default: the number of available cores in the machine - 1)
file.prefix	A character string that will be used as a prefix followed by "_ROCR_perf.png" for the result plot file, if it is NULL (default), no plot is returned
kernel	SVM kernel, a character string: "linear" or "radial". (default = "radial")
scale	Logical indicating if the data have to be scaled or not (default = FALSE)
cost	The SVM cost parameter for both linear and radial kernels. If NULL (default), the function mcTune is run.
gamma	The SVM gamma parameter for radial kernel. If radial kernel and NULL (default), the function mcTune is run.

Value

A list with two objects.

probs	The predictions computed by the model for each subset during the cross-validation
labels	The actual class for each subset

Examples

```
data(crm.features)
data(feature.ranking)
#probs.labels.list <- evaluateModelPerformance(data.granges=crm.features,
#    feature.ranking=feature.ranking, feature.nb=50,
#    file.prefix = "test")
#names(probs.labels.list[[1]])
```

---

feature.matrix *This is data to be included in my package*

Description

This is data to be included in my package

---

feature.ranking *This is data to be included in my package*

Description

This is data to be included in my package

LedPred *Creates an SVM model given a feature matrix*

Description

The LedPred function computes the best SVM parameters, defines the optimal features for creating the SVM model by running sequentially mcTune, rankFeatures, tuneFeatureNb and createModel. The performances of this model are then computed usong evaluateModelPerformance.

Usage

```
LedPred(data = NULL, data.granges = NULL, cl = 1, ranges = list(gamma =
  seq(from = 1, to = 10, by = 9), cost = seq(from = 1, to = 10, by = 9)),
  kernel = "linear", scale = FALSE, valid.times = 10,
  file.prefix = NULL, numcores = parallel::detectCores() - 1,
  step.nb = 20, halve.above = 100)
```

Arguments

data	data.frame containing the training set
data.granges	Bioconductor GenomicRanges object containing the training set
cl	integer indicating the column number corresponding to the response vector that classify positive and negative regions (default = 1)
ranges	list object containing one (linear kernel) or two (radial kernel) vectors of integers corresponding to SVM cost and SVM gamma parameters to test.
kernel	SVM kernel, a character string: "linear" or "radial". (default = "radial")
scale	Logical indicating if the data have to be scaled or not (default = FALSE)
valid.times	Integer indicating how many times the training set will be split for the cross validation step (default = 10). This number must be smaller than positive and negative sets sizes.
file.prefix	A character string that will be used as a prefix for the result files. If it is NULL (default), no plot is returned
numcores	Number of cores to use for parallel computing (default: the number of available cores in the machine - 1)
step.nb	Number of features to add at each step (default = 10)
halve.above	During RFE, all the features are ranked at the first round and the half lowest ranked features (that contribute the least in the model) are removed for the next round. When the number of feauture is lower or equal to halve.above, the features are removed one by one. (default=100)

Value

A list of the object produced at each step

`best.params`	A list of the parameters giving the lowest misclassification error
`feature.ranking`	List of ordered features from `rankFeatures`
`feature.nb`	he optimal number of feature to use from the list of ordered features from `tuneFeatureNb`
`model.svm`	The best SVM model `createModel`
`probs.label.list`	The cross-validation results from `evaluateModelPerformance`

Examples

```
data(crm.features)
#cost_vector <- c(1,3,10)
#gamma_vector <- c(1,3,10)
#ledpred.list=LedPred(data.granges=crm.features, cl=1, ranges = list(cost=cost_vector,
#                          gamma=gamma_vector), kernel="linear", halve.above=50)
#names(ledpred.list)
```

`mapFeaturesToCRMs` *R interface to bed_to_matrix REST in server*

Description

The `mapFeaturesToCRMs` function allows the user to create a training set matrix to build a predictive model. The training set is composed of positive regions (known to be involved in the pathway of interest) and negative regions (randomly picked or known to not be involved in the pathway of interest) that will be described (scored) by features. Three types of features file format are accepted: Position specific scoring matrices modeling motifs recognised by transcription factors, bed files containing region coordinates for any discrete feature (NGS peaks, conservation blocks) and wig/bigWig files containing signal data. This script has been tested with version 0.99 of the online server. Go here to see current version of the server http://ifbprod.aitorgonzalezlab.org/map_features_to_crms.php

Usage

```
mapFeaturesToCRMs(URL = "http://ifbprod.aitorgonzalezlab.org/map_features_to_crms.php",
 positive.bed = NULL, genome = NULL, negative.bed = NULL,
 shuffling = NULL, background.seqs = NULL, genome.info = NULL,
 pssm = NULL, background.freqs = NULL, ngs = NULL, bed.overlap = NULL,
 my.values = NULL, feature.ranking = NULL, feature.nb = NULL,
 crm.feature.file = NULL, stderr.log.file = NULL, stdout.log.file = NULL)
```

Arguments

URL	URL of the server REST target
positive.bed	Positive bed file path. Compulsory
genome	Genome code, eg. dm3 for Drosophila Melanogaster. Compulsory
negative.bed	Negative bed file path.
shuffling	Integer with number of time shuffle background sequences (background.seqs). If negative.bed is NULL and shuffling is set at 0, the feature matrix does not contain negative sequences. It is useful to produce a test set matrix.
background.seqs	Background sequences used for shuffling. If shuffling = 0, set this parameter at 0.
genome.info	File require for shuffling bed. If shuffling = 0, set this parameter at 0.
pssm	Position specific scoring matrices
background.freqs	Background frequencies of nucleotides in genome
ngs	NGS (bed and wig) files
bed.overlap	Minimal overlap as a fraction of query sequence with NGS bed peak. Equivalent with intersectBed -f argument. Default 1bp.
my.values	Bed file where fourth column are values to append to the SVM matrix
feature.ranking	File with ranked features (Output of rankFeatures). It is used for scoring a query bed file
feature.nb	Integer with feature.nb
crm.feature.file	Path to feature matrix file
stderr.log.file	Path to error log
stdout.log.file	Path to standard output log

Value

A list

feature.matrix	a data frame where each row is a region and each column a feature, each cell carry a score, the first column is the response vector
stdout.log	Standard output log of mapFeaturesToCRMs script in server
stderr.log	Standard error log of mapFeaturesToCRMs script in server

Examples

```
## Not run:
 dirPath <- system.file("extdata", package="LedPred")
 file.list <-   list.files(dirPath, full.names=TRUE)
 background.freqs <- file.list[grep("freq", file.list)]
 positive.regions <-  file.list[grep("positive", file.list)]
 negative.regions <-  file.list[grep("negative", file.list)]
 TF.matrices <-  file.list[grep("tf", file.list)]
 ngs.path <- system.file("extdata/ngs", package="LedPred")
 ngs.files=list.files(ngs.path, full.names=TRUE)
 crm.features.list <- mapFeaturesToCRMs(positive.bed=positive.regions,
     negative.bed=negative.regions,  background.freqs=background.freqs,
     pssm=TF.matrices, genome="dm3", ngs=ngs.files,
     crm.feature.file = "crm.features.tab",
     stderr.log.file = "stderr.log", stdout.log.file = "stdout.log")
 names(crm.features.list)
 class(crm.features.list$crm.features)
 crm.features.list$stdout.log
 crm.features.list$stderr.log

## End(Not run)
```

`mcTune` *Tuning the SVM parameters*

Description

The `mcTune` function is a modified version of the function `tune` from package e1071 [6]. It tests the different combinations of C and gamma parameters given as vectors in a list and will return the prediction error computed during the cross-validation step.

Usage

```
mcTune(data = NULL, data.granges = NULL, cl = 1, ranges = list(gamma =
  c(1, 10), cost = c(1, 10)), kernel = "linear", scale = FALSE,
  valid.times = 10, file.prefix = NULL, numcores = parallel::detectCores()
  - 1)
```

Arguments

`data`	data.frame containing the training set
`data.granges`	Bioconductor GenomicRanges object containing the training set
`cl`	integer indicating the column number corresponding to the response vector that classify positive and negative regions (default = 1)
`ranges`	list object containing one (linear kernel) or two (radial kernel) vectors of integers corresponding to SVM cost and SVM gamma parameters to test.
`kernel`	SVM kernel, a character string: "linear" or "radial". (default = "radial")

scale	Logical indicating if the data have to be scaled or not (default = FALSE)
valid.times	Integer indicating how many times the training set will be split for the cross validation step (default = 10). This number must be smaller than positive and negative sets sizes.
file.prefix	A character string that will be used as a prefix followed by "_c_g_eval.png" for result plot files, if it is NULL (default), no plot is returned
numcores	Number of cores to use for parallel computing (default: the number of available cores in the machine - 1)

Value

A list of class tune

best.parameters	A list of the parameters giving the lowest misclassification error
best.performance	The lowest misclassification error
method	The method used
nparcomb	the number of tested parameter combinations
train.ind	The indexes used to produce subsets during the cross validation step
sampling	The cross-validation fold number
performances	A matrix summarizing the cross-validation step with the error for each tested parameter at each round and the dispersion of these errors (regarding to the average error)
best.model	The model produced by the best parameters

Examples

```
data(crm.features)
cost.vector <- c(1,3,10,30)
gamma.vector <- c(1,3,10,30)
#c.g.obj <- mcTune(data.granges= crm.features, ranges = list(cost=cost.vector,
#    gamma=gamma.vector), kernel='linear', file.prefix = "test")
#names(c.g.obj)
# cost <- c.g.obj$best.parameters$cost
# gamma <- c.g.obj$best.parameters$gamma
```

---

rankFeatures *Ranking the features according to their importance*

Description

The rankFeatures function performs a Recursive Feature Elimination (RFE) on subsets of the feature matrix. For each subset the features are ranked according to the weight attributed by SVM at each round of elimination and the average rank of each feature over the subsets is returned. We recommand to save the object containing the ranked features for the following steps.

Usage

```
rankFeatures(data = NULL, data.granges = NULL, cl = 1,
  halve.above = 100, valid.times = 10, kernel = "linear", cost = 1,
  gamma = 1, scale = FALSE, numcores = parallel::detectCores() - 1,
  file.prefix = NULL)
```

Arguments

data	data.frame containing the training set
data.granges	Bioconductor GenomicRanges object containing the training set
cl	integer indicating the column number corresponding to the response vector that classify positive and negative regions (default = 1)
halve.above	During RFE, all the features are ranked at the first round and the half lowest ranked features (that contribute the least in the model) are removed for the next round. When the number of feauture is lower or equal to halve.above, the features are removed one by one. (default=100)
valid.times	Integer indicating how many times the training set will be split (default = 10). This number must be smaller than positive and negative sets sizes.
kernel	SVM kernel, a character string: "linear" or "radial". (default = "radial")
cost	The SVM cost parameter for both linear and radial kernels. If NULL (default), the function mcTune is run.
gamma	The SVM gamma parameter for radial kernel. If radial kernel and NULL (default), the function mcTune is run.
scale	Logical indicating if the data have to be scaled or not (default = FALSE)
numcores	Number of cores to use for parallel computing (default: the number of available cores in the machine - 1)
file.prefix	A character string that will be used as a prefix for output file, if it is NULL (default), no file is writen.

Value

A 3-columns data frame with ranked features. First column contains the feature names, the second the original position of the feature in the feature.matrix and the third the average rank over the subsets.

Examples

```
data(crm.features)
cost <- 1
gamma <- 1
 #feature.ranking <- rankFeatures(data.granges=crm.features, cost=cost,gamma=gamma,
 #    kernel='linear', file.prefix = "test", halve.above=10)
```

scoreData *Predicting new regulatory regions*

Description

scoreData function predict new regulatory regions using SVM model from a test data set

Usage

```
scoreData(data = NULL, data.granges = NULL, model, score.file = NULL,
  score.bed.file = NULL)
```

Arguments

data	data.frame containing the test set. This test set must have the same descriptive features as the one that were used to build the model.
data.granges	Bioconductor GenomicRanges object containing the test set
model	the SVM model
score.file	A character string that will be used as the file name for the output file, if it is NULL (default), no file is writen. The output file takes the form of two columns with object names and scores.
score.bed.file	A character string that will be used as the file name for the output bed file, if it is NULL (default), no bed file is writen

Value

A 2-columns dataframe. First column containg the SVM model prediction probabilities and the second containing the corresponding regions

Examples

```
data(crm.features)
data(svm.model)
#pred.test <- scoreData(data.granges=crm.features, model=svm.model,
# score.file="test_prediction.tab")
```

svm.model *This is data to be included in my package*

Description

This is data to be included in my package

`tuneFeatureNb` *Selecting the optimal number of features*

Description

`tuneFeatureNb` iterates through increasing feature numbers to calculate kappa values which represents the performance of the model computed with the given features. We recommand to save the object containing the optimal number of features for the following steps.

Usage

```
tuneFeatureNb(data = NULL, data.granges = NULL, feature.ranking = NULL,
  cl = 1, valid.times = 10, cost = 1, gamma = 1, kernel = "linear",
  scale = FALSE, step.nb = 10, numcores = parallel::detectCores() - 1,
  file.prefix = NULL)
```

Arguments

`data`	data.frame containing the training set
`data.granges`	Bioconductor GenomicRanges object containing the training set
`feature.ranking`	List of ordered features.
`cl`	integer indicating the column number corresponding to the response vector that classify positive and negative regions (default = 1)
`valid.times`	Integer indicating how many times the training set will be split for the cross validation step (default = 10). This number must be smaller than positive and negative sets sizes.
`cost`	The SVM cost parameter for both linear and radial kernels. If NULL (default), the function `mcTune` is run.
`gamma`	The SVM gamma parameter for radial kernel. If radial kernel and NULL (default), the function `mcTune` is run.
`kernel`	SVM kernel, a character string: "linear" or "radial". (default = "radial")
`scale`	Logical indicating if the data have to be scaled or not (default = FALSE)
`step.nb`	Number of features to add at each step (default = 10)
`numcores`	Number of cores to use for parallel computing (default: the number of available cores in the machine - 1)
`file.prefix`	A character string that will be used as a prefix followed by "_kappa_measures.png" for the result plot file. If it is NULL (default), no plot is returned

Value

A list with two objects.

`performance` 2-columns data frame. first column correspond to the number of tested features, second column contains the corresponding kappa value

`best.feature.nb` Integer corresponding to the number of features producing the model with the highest kappa value

Examples

```
data(crm.features)
data(feature.ranking)
cost <- 1
gamma <- 1
 #feature.nb.obj <- tuneFeatureNb(data.granges=crm.features,
 #    feature.ranking=feature.ranking, kernel='linear', cost=cost,gamma=gamma,
 #    file.prefix = "test")
 #names(feature.nb.obj)
```

Index