Package 'UBayFS'

January 20, 2025

Type Package

Title A User-Guided Bayesian Framework for Ensemble Feature Selection

Version 1.0

Description The framework proposed in Jenul et al., (2022) <doi:10.1007/s10994-022-06221-9>, together with an interactive Shiny dashboard. 'UBayFS' is an ensemble feature selection technique embedded in a Bayesian statistical framework. The method combines data and user knowledge, where the first is extracted via data-driven ensemble feature selection. The user can control the feature selection by assigning prior weights to features and penalizing specific feature combinations. 'UBayFS' can be used for common feature selection as well as block feature selection.

License GPL-3

URL https://annajenul.github.io/UBayFS/,
https://joss.theoj.org/papers/10.21105/joss.04848

BugReports https://github.com/annajenul/UBayFS/issues

Depends R (>= 3.5.0)

Imports DirichletReg, GA, ggplot2, gridExtra, hyper2, matrixStats, methods, mRMRe, Rdimtools, shiny, utils

Suggests caret, dplyr, DT, glmnet, GSelection, knitr, plyr, RColorBrewer, rmarkdown, rpart, shinyalert, shinyBS, shinyjs, shinyWidgets, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

NeedsCompilation no

Author Anna Jenul [aut, cre] (<https://orcid.org/0000-0002-6919-3483>), Stefan Schrunner [aut] (<https://orcid.org/0000-0003-1327-4855>), Kristian Hovde Liland [rev], Oliver Tomic [ctb], Jürgen Pilz [ctb]

Maintainer Anna Jenul <anna.jenul@nmbu.no>

Repository CRAN

Date/Publication 2023-03-07 10:50:02 UTC

Contents

---

bcw *Breast Cancer Wisconsin dataset*

Description

A dataset containing features computed from digitized images of a fine needle aspirate (FNA) of a breast mass. The target function contains two classes representing patient diagnoses (M...malignant and B...benign). The dataset has been taken from the UCI Repository of Machine Learning Databases and was created by W. H. Wolberg, W. N. Street and O. L. Mangasarian in 1995. For details, see UCI documentation or literature:

- doi:10.1117/12.148698
- https://www.jstor.org/stable/171686

Feature blocks were added to the original dataset according to the dataset description (10 blocks corresponding to different image characteristics).

Usage

```
bcw
```

Format

A list containing:

- a matrix ‘data‘ with 569 rows and 30 columns representing features,
- a vector ‘labels‘ of factor type with 569 entries representing the binary target variable, and
- a list of feature indices representing feature blocks.

Source

https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)

---

build.UBayconstraint *Build a customized constraint for UBayFS*

Description

Builds a constraint using a left side ‘A‘, a right side ‘b‘, a relaxation parameter ‘rho‘, and a block matrix ‘block_matrix‘.

Usage

```
build.UBayconstraint(A, b, rho, block_matrix = NULL)
```

Arguments

A	matrix containing the left side of the linear inequality system
b	vector containing the right side of the linear inequality system
rho	vector containing the relaxation parameters for each constraint
block_matrix	a matrix indicating the membership of features in feature blocks

Value

a ‘UBayconstraint‘ object

`build.UBaymodel` *Build an ensemble for UBayFS*

Description

Build a data structure for UBayFS and train an ensemble of elementary feature selectors.

Usage

```
build.UBaymodel(
  data,
  target,
  M = 100,
  tt_split = 0.75,
  nr_features = "auto",
  method = "mRMR",
  prior_model = "dirichlet",
  weights = 1,
  constraints = NULL,
  lambda = 1,
  optim_method = "GA",
  popsize = 50,
  maxiter = 100,
  shiny = FALSE,
  ...
)
```

Arguments

`data`	a matrix of input data
`target`	a vector of input labels; for binary problems a factor variable should be used
`M`	the number of elementary models to be trained in the ensemble
`tt_split`	the ratio of samples drawn for building an elementary model (train-test-split)
`nr_features`	number of features to select in each elementary model; if 'auto' a randomized number of features is used in each elementary model
`method`	a vector denoting the method(s) used as elementary models; options: 'mRMR', 'laplace' (Laplacian score) Also self-defined functions are possible methods; they must have the arguments X (data), y (target), n (number of features) and name (name of the function). For more details see examples.
`prior_model`	a string denoting the prior model to use; options: 'dirichlet', 'wong', 'hankin'; 'hankin' is the most general prior model, but also the most time consuming
`weights`	the vector of user-defined prior weights for each feature
`constraints`	a list containing a relaxed system 'Ax<=b' of user constraints, given as matrix 'A', vector 'b' and vector or scalar 'rho' (relaxation parameter). At least one max-size constraint must be contained. For details, see buildConstraints.

lambda	a positive scalar denoting the overall strength of the constraints
optim_method	the method to evaluate the posterior distribution. Currently, only the option 'GA' (genetic algorithm) is supported.
popsize	size of the initial population of the genetic algorithm for model optimization
maxiter	maximum number of iterations of the genetic algorithm for model optimization
shiny	TRUE indicates that the function is called from Shiny dashboard
...	additional arguments

Details

The function aggregates input parameters for UBayFS - including data, parameters defining ensemble and user knowledge and parameters specifying the optimization procedure - and trains the ensemble model.

Value

a 'UBaymodel' object containing the following list elements:

- 'data' - the input dataset
- 'target' - the input target
- 'lambda' - the input lambda value (constraint strength)
- 'prior_model' - the chosen prior model
- 'ensemble.params' - information about input and output of ensemble feature selection
- 'constraint.params' - parameters representing the constraints
- 'user.params' - parameters representing the user's prior knowledge
- 'optim.params' - optimization parameters

Examples

```
# build a UBayFS model using Breast Cancer Wisconsin dataset
data(bcw) # dataset
c <- buildConstraints(constraint_types = 'max_size',
                      constraint_vars = list(10),
                      num_elements = ncol(bcw$data),
                      rho = 1) # prior constraints
w <- rep(1, ncol(bcw$data)) # weights
model <- build.UBaymodel(
                     data = bcw$data,
                     target = bcw$labels,
                     M = 20,
                     constraints = c,
                     weights = w
)

# use a function computing a decision tree as input
library('rpart')
decision_tree <- function(X, y, n, name = 'tree'){
```

```
rf_data = as.data.frame(cbind(y, X))
colnames(rf_data) <- make.names(colnames(rf_data))
tree = rpart::rpart(y~., data = rf_data)
return(list(ranks= which(colnames(X) %in% names(tree$variable.importance)[1:n]),
           name = name))
}

model <- build.UBaymodel(
                     data = bcw$data,
                     target = bcw$labels,
                     constraints = c,
                     weights = w,
                     method = decision_tree
)

# include block-constraints
c_block <- buildConstraints(constraint_types = 'max_size',
                            constraint_vars = list(2),
                            num_elements = length(bcw$blocks),
                            rho = 10,
                            block_list = bcw$blocks)

model <- setConstraints(model, c_block)
```

`buildConstraints` *Build a constraint system*

Description

Build an inequation system from constraints provided by the user.

Usage

```
buildConstraints(
  constraint_types,
  constraint_vars,
  num_elements,
  rho = 1,
  block_list = NULL,
  block_matrix = NULL
)
```

Arguments

`constraint_types`
: a vector of strings denoting the type of constraint to be added; options: ‘max_size‘, ‘must_link‘, ‘cannot_link‘

`constraint_vars`
: a list of parameters defining the constraints; in case of max-size constraints, the list element must contain an integer denoting the maximum size of the feature set, in case of max-link or cannot link, the list element must be a vector of feature indices to be linked

`num_elements`
: the total number of features (feature-wise constraints) or blocks (block-wise constraints) in the dataset

`rho`
: a positive parameter denoting the level of relaxation; 'Inf' denotes a hard constraint, i.e. no relaxation

`block_list`
: the list of feature indices for each block; only required, if block-wise constraints are built and 'block_matrix' is 'NULL'

`block_matrix`
: the matrix containing affiliations of features to each block; only required, if block-wise constraints are built and 'block_list' is 'NULL'

Details

The function transforms user information about relations between features (must-link or cannot-link constraints) and maximum feature set size (max-size) into a linear inequation system. In addition, the relaxation parameter 'rho' can be specified to achieve soft constraints.

Value

a 'UBayconstraint' containing a matrix 'A' and a vector 'b' representing the inequality system 'Ax<=b', and a vector 'rho' representing the penalty shape

Examples

```
# given a dataset with 10 features, we create a max-size constraint limiting
# the set to 5 features and a cannot-link constraint between features 1 and 2
buildConstraints(constraint_types = c('max_size','cannot_link'),
                 constraint_vars = list(5, c(1,2)),
                 num_elements = 10,
                 rho = 1)
```

---

buildDecorrConstraints

Build decorrelation constraints

Description

Build a cannot link constraint between highly correlated features. The user defines the correlation threshold.

Usage

```
buildDecorrConstraints(data, level = 0.5, method = "spearman")
```

Arguments

data	the dataset in the ‘UBaymodel‘ object
level	the threshold correlation-level
method	the method used to compute correlation; must be one of ‘pearson‘, ‘spearman‘ or ‘kendall‘

Value

a list containing a matrix ‘A‘ and a vector ‘b‘ representing the inequality system ‘Ax<=b‘, a vector ‘rho‘ and a block matrix

---

build_train_set *Perform stratified data partition.*

Description

Sample indices for training from the data.

Usage

```
build_train_set(y, tt_split)
```

Arguments

y	a column, often the target, by which the data shall be partitioned.
tt_split	the percentage of data used for training in each ensemble model.

Value

data indices for training ensembles

---

evaluateFS *Evaluate a feature set*

Description

Evaluates a feature set under the UBayFS model framework.

Usage

```
evaluateFS(state, model, method = "spearman", log = FALSE)

evaluateMultiple(state, model, method = "spearman", log = TRUE)
```

Arguments

`state`	a binary membership vector describing a feature set
`model`	a 'UBaymodel' object created using build.UBaymodel
`method`	type of correlation ('pearson','kendall', or 'spearman')
`log`	whether the admissibility should be returned on log scale

Value

a posterior probability value

Functions

- `evaluateMultiple()`: Evaluate multiple feature sets

`group_admissibility` *Admissibility for constraint group*

Description

Evaluate the value of the admissibility function 'kappa'.

Usage

```
group_admissibility(state, constraints, log = TRUE)

admissibility(state, constraint_list, log = TRUE)
```

Arguments

`state`	a binary membership vector describing a feature set
`constraints`	group of constraints with common block matrix
`log`	whether the admissibility should be returned on log scale
`constraint_list`	a list of constraint groups, each containing a matrix 'A' and a vector 'b' representing the inequality system 'Ax<=b', a vector 'rho', and a matrix 'block_matrix'

Value

an admissibility value

Functions

- `group_admissibility()`: computes admissibility for a group of constraints (with a common block).

is.UBayconstraint *Checks whether a list object implements proper UBayFS user constraints*

Description

Checks whether a list object implements proper UBayFS user constraints

Usage

```
is.UBayconstraint(x)
```

Arguments

x	a ‘UBayconstraint‘ object

Value

boolean value

is.UBaymodel *Check whether an object is a UBaymodel*

Description

Perform consistency checks of a UBaymodel.

Usage

```
is.UBaymodel(x)
```

Arguments

x	an object to be checked for class consistency

Value

returns a single scalar (TRUE or FALSE) indicating whether the object fulfills the consistency requirements of the UBayFS model

posteriorExpectation *Posterior expectation of features*

Description

compute the posterior score for each feature.

Usage

```
posteriorExpectation(model)
```

Arguments

model	a ‘UBaymodel‘ object

print.UBayconstraint *Prints the ‘UBayconstraint‘ object*

Description

Prints the ‘UBayconstraint‘ object

Usage

```
## S3 method for class 'UBayconstraint'
print(x, ...)

## S3 method for class 'UBayconstraint'
summary(object, ...)
```

Arguments

x	a ‘UBayconstraint‘ object
...	additional print parameters
object	a ‘UBayconstraint‘ object

Value

prints model summary to the console, no return value

Functions

- summary(UBayconstraint): Prints a summary of the ‘UBayconstraint‘ object

`print.UBaymodel` *Print a UBayFS model*

Description

Print details of a 'UBaymodel'

Usage

```
## S3 method for class 'UBaymodel'
print(x, ...)

printResults(model)

## S3 method for class 'UBaymodel'
summary(object, ...)

## S3 method for class 'UBaymodel'
plot(x, ...)
```

Arguments

`x`	a 'UBaymodel' object created using build.UBaymodel
`...`	additional print parameters
`model`	a 'UBaymodel' object created using build.UBaymodel after training
`object`	a 'UBaymodel' object created using build.UBaymodel

Value

prints model summary to the console, no return value

Functions

- `printResults()`: Display and summarize the results of UBayFS after feature selection.
- `summary(UBaymodel)`: A summary of a 'UBaymodel'
- `plot(UBaymodel)`: A barplot of a 'UBaymodel' containing prior weights, ensemble counts and the selected features.

runInteractive *Run an interactive Shiny app for demonstration*

Description

Starts an interactive R Shiny application in the browser.

Usage

```
runInteractive()
```

Value

calls Shiny app, no return value

sampleInitial *Initial feature set sampling using probabilistic Greedy algorithm*

Description

Sample initial solutions using a probabilistic version of Greedy algorithm.

Usage

```
sampleInitial(post_scores, constraints, size)
```

Arguments

post_scores	a vector of posterior scores (prior scores + likelihood) for each feature
constraints	a list containing feature-wise constraints
size	initial number of samples to be created. The output sample size can be lower, since duplicates are removed.

Value

a matrix containing initial feature sets as rows

setConstraints *Set constraints in UBaymodel object*

Description

Set the constraints in a 'UBaymodel' object.

Usage

```
setConstraints(model, constraints, append = FALSE)
```

Arguments

model	a 'UBaymodel' object created using build.UBaymodel
constraints	a 'UBayconstraint' object created using build.UBayconstraint
append	if 'TRUE', constraints are appended to the existing constraint system

Value

a 'UBaymodel' object with updated constraint parameters

See Also

build.UBaymodel

setOptim *Set optimization parameters in a UBaymodel object*

Description

Set the optimization parameters in a UBaymodel object.

Usage

```
setOptim(model, method = "GA", popsize, maxiter)
```

Arguments

model	a UBaymodel object created using build.UBaymodel
method	the method to evaluate the posterior distribution; currently only'GA' (genetic algorithm) is supported
popsize	size of the initial population of the genetic algorithm for model optimization
maxiter	maximum number of iterations of the genetic algorithm for model optimization

Value

a UBaymodel object with updated optimization parameters

See Also

build.UBaymodel

`setWeights` *Set weights in UBaymodel object*

Description

Set the prior weights in a UBaymodel object.

Usage

```
setWeights(model, weights, block_list = NULL, block_matrix = NULL)
```

Arguments

`model`	a UBaymodel object created using build.UBaymodel
`weights`	the vector of user-defined prior weights for each feature
`block_list`	the list of feature indices for each block; only required, if block-wise weights are specified and block_matrix is NULL
`block_matrix`	the matrix containing affiliations of features to each block; only required, if block-wise weights are specified and block_list is NULL

Value

a UBaymodel object with updated prior weights

See Also

build.UBaymodel

`train` *UBayFS feature selection*

Description

Genetic algorithm to train UBayFS feature selection model.

Usage

```
train(x, verbose = FALSE)
```

Arguments

`x`	a 'UBaymodel' created by build.UBaymodel
`verbose`	if TRUE: GA optimization output is printed to the console

Value

a 'UBaymodel' with an additional list element output containing the optimized solution, see build.UBaymodel

Index