Package: bsnsing (via r-universe)

October 18, 2024

Type Package

Title Bsnsing: A Decision Tree Induction Method Based on Recursive Optimal Boolean Rule Composition

Version 1.0.1

Author Yanchao Liu

Maintainer Yanchao Liu <yanchaoliu@wayne.edu>

Description The bsnsing package provides functions for training a decision tree classifier, making predictions and generating latex code for plotting. It solves the two-class and multi-class classification problems under the supervised learning paradigm. While building a decision tree, bsnsing uses a Boolean rule involving multiple variables to split a node. Each split rule is identified by solving an optimization problem. Use the bsnsing function to build a tree, the predict function to make predictions and the show function to plot the tree. The paper is at <arXiv:2205.15263>. Source code and more data sets are at <https://github.com/profyliu/bsnsing>.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Imports lpSolve, methods

Suggests slam, cplexAPI, gurobi, C50, party, rpart, tree

Repository https://profyliu.r-universe.dev

RemoteUrl https://github.com/profyliu/bsnsing

RemoteRef HEAD

RemoteSha c6abcc9f4e2f02a01192dddbaef5a5668a84326f

Contents

`bsnsing-package` *bsnsing: A decision tree induction method based on recursive optimal boolean rule composition*

Description

The bsnsing package provides functions for building a decision tree classifier and making predictions. It solves a mixed-integer programming (MIP) model to maximize the Gini reduction at each node split, and each node split rule can utilize multiple input variables. Benchmarking experiments on 75 open data sets suggest that bsnsing trees are the most capable of discriminating new cases compared to trees trained by other decision tree codes including the rpart, C50, party and tree packages in R. Compared to other optimal decision tree packages, including DL8.5, OSDT, GOSDT and indirectly more, bsnsing stands out in its training speed, ease of use and broader applicability without losing in prediction accuracy. For more information, please check out the paper `https://arxiv.org/abs/2205.15263`, to be published in INFORMS Journal on Computing.

The ENUM algorithm

The default method for solving the MIP model is the implicit enumeration (ENUM) algorithm, while other solvers including GUROBI, CPLEX and lpSolve can be used (via specifying the opt.solver option in the `bsnsing` function). However, the users are strongly suggested to compile the bslearn.c file, make it into a shared library (e.g., .dylib, .so or .dll binary file) and paste the binary file in the work directory. In this way, the bsnsing will leverage the compiled code (instead of the R code) for the ENUM algorithm, which runs much (~40x) faster. All benchmarking experiments were run using the compiled ENUM algorithm. The C source file and the MAKE file can be found at `https://github.com/profyliu/bsnsing`. Pre-compiled binary files for different target platforms are also provided there for the convenience of the users (just download the .dylib, .so or the .dll file, depending on the operating system, and put it in the work directory). Future updates of this package will internalize the compilation step, but for now only the R implementation of the ENUM algorithm is included in the package source, so serious users please take the extra step outlined above.

More data sets

Several data frames (i.e., `auto`, `iris`, `GlaucomaMVF` and `BreastCancer`) used in the example code are included in this package. More two-class and multi-class classification data sets can be found at `https://github.com/profyliu/bsnsing`.

Learn functions

The learn (train) functions include `bsnsing`, `bsnsing.formula` and `bsnsing.default`.

Predict functions

The predict functions include: `predict.bsnsing` and `predict.mbsnsing`.

Plot functions

A `bsnsing` object (tree) can be plotted into a PDF file, or in the form of latex code, by the function `show.bsnsing`. The ROC curve can be plotted using the function `ROC_func`.

Auxilliary functions

Here is a list of internal functions of the package that are open for end users. `summary.bsnsing` `summary.mbsnsing` `binarize`, `binarize.numeric`, `binarize.factor`, `binarize.y`, `bslearn`, `bscontrol`

Author(s)

Yanchao Liu

`auto` *auto*

Description

A test data set.

Usage

```
auto
```

Format

A data frame with 392 rows and 8 variables

Source

http://github.com/profyliu/bsnsing

`binarize` *Create Binary Variables by the Classification Target*

Description

Create a set of variables (columns) with binary values for each column in the input data. For a variable with values of 0 and 1, the column is retained and no new column is created. For a numeric variable, the function `binarize.numeric` is called. For a factor column, the function `binarize.factor` is called.

Usage

```
binarize(
  x,
  y,
  target = stop("'target' (0 or 1) must be provided"),
  control = bscontrol()
)
```

Arguments

`x`	a data frame or matrix to be binarized.
`y`	a vector with two unique values (0 and 1). It is the response variable that guides the optimal discretization of variables in `x`.
`target`	the level of `y` (0 or 1) which indicates the boolean rule target

`control`	a list or a `bscontrol()` object. The list should contain the following three attributes: *nseg.numeric*, a positive integer indicating the maximum number of segments used in discretizing a numeric variable, *nseg.factor*, a positive integer indicating the maximum number of levels allowed for a factor variable, and *bin.size*, a positive integer indicating the minimum number of observations to fall in a segment.

Value

a data frame containing binary variables, or a character string describing the rule that perfectly split the target.

Examples

```
## Not run:
# Load and prepare data
x <- auto[, c('mpg', 'cylinders', 'displacement')]
x$cylinders <- as.factor(x$cylinders)
y <- ifelse(auto$origin == 'USA', 1L, 0L)
# binarize x by y = 1
bx1 <- binarize(x, y, target = 1)
head(bx1)
# binarize x by y = 0
bx0 <- binarize(x, y, target = 0)
head(bx0)
# when selecting only one column from a data frame, use drop = FALSE to maintain
binarize(auto[,'mpg', drop = FALSE], y, target = 1)

## End(Not run)
```

`binarize.factor` *Create Binary Features based on a Factor Vector*

Description

Create binary dummy variables based on a factor variable. This function is used internally by `binarize`.

Usage

```
binarize.factor(x, name, y, segments = 10, bin.size = 5)
```

Arguments

`x`	a numeric vector.
`name`	a character string, the variable name of `x`.
`y`	a numeric or integer vector of the same length as `x`, consisting of two unique values: 0 and 1.

`segments`	a positive integer indicating the maximum number of levels allowed in the factor variable.
`bin.size`	a positive integer. It is the minimum number of observations required to fall into each bin.

Value

a data frame with binary (0 and 1) entries. The column names are indicative of the conditions used to form the corresponding columns.

`binarize.numeric` *Create Binary Features based on a Numeric Vector*

Description

Discretize a continuous variable `x` by splitting its range at a sequence of cutpoints. The cutpoints are determined so as to effectively split the binary target `y`. This function is used internally by `binarize`.

Usage

```
binarize.numeric(
  x,
  name,
  y,
  target = stop("Must provide a target, 0 or 1"),
  segments = 10,
  bin.size = 5,
  node.size = 10
)
```

Arguments

`x`	a numeric vector.
`name`	a character string, the variable name of `x`.
`y`	a numeric or integer vector of the same length as `x`, consisting of two unique values: 0 and 1.
`target`	a scalar, valued 0 or 1, indicating the target level of `y`.
`segments`	a positive integer, any value below 3 is set to 3. It is the maximum number of segments the range of `x` is divided into.
`bin.size`	a positive integer. It is the minimum number of observations required to fall into each bin.
`node.size`	a positive integer. If either child node is smaller than the node.size, do not return the perfect rule.

Value

a data frame with binary (0 and 1) entries, or a character string describing the rule that perfectly splits `y`. If a data frame is returned, the column names are indicative of the conditions used to form the corresponding columns.

`binarize.y` *Recode a Variable with Two Unique Values into an 0/1 Vector*

Description

Recode a Variable with Two Unique Values into an 0/1 Vector

Usage

```
binarize.y(y, verbose = F)
```

Arguments

`y`	a vector, must contain two unique values.
`verbose`	a logical value, TRUE or FALSE, indicating whehter details are to be printed on the screen.

Value

a list with three elements: `y`, a vector of the same length as `y`, whose entries are coded to 0 and 1, `coding.scheme`, a character string describing the map from the original coding to 0/1 coding, and `ycode`, a character vector containing the original level names of `y`.

Examples

```
y <- factor(c('good', 'bad', 'good', 'good', 'bad'))
(yb <- binarize.y(y))
y <- c(TRUE, FALSE, FALSE, FALSE, TRUE)
(yb <- binarize.y(y))
y <- c(1, 2, 2, 1, 2)
(yb <- binarize.y(y))
```

BreastCancer *BreastCancer*

Description

A test data set.

Usage

```
BreastCancer
```

Format

A data frame with 699 rows and 10 variables.

Source

http://github.com/profyliu/bsnsing

bscontrol *Define Parameters for the bsnsing Fit*

Description

Define Parameters for the bsnsing Fit

Usage

```
bscontrol(
  bin.size = 5,
  nseg.numeric = 20,
  nseg.factor = 20,
  num2factor = 10,
  node.size = 0,
  stop.prob = 0.9999,
 opt.solver = c("enum_c", "enum", "greedy", "hybrid", "gurobi", "lpSolve", "cplex"),
  solver.timelimit = 180,
  max.rules = 2,
  opt.model = c("gini", "error"),
  greedy.level = 0.9,
  import.external = T,
  suppress.internal = F,
  no.same.gender.children = F,
  n0n1.cap = 40000,
  verbose = F
)
```

Arguments

`bin.size`	the minimum number of observations required in a binarization bucket.
`nseg.numeric`	the maximum number of segments the range of a numeric variable is divided into for each inequality direction.
`nseg.factor`	the maximum number of unique levels allowed in a factor variable.
`num2factor`	an equality binarization rule will be created for each unique value of a numeric variable (in addition to the inequality binarization attempt), if the number of unique values of the numeric variable is less than `num2factor`.
`node.size`	if the number of training cases falling into a tree node is fewer than `node.size`, the node will become a leaf and no further split will be attempted on it; in addition, do not split a node if either child node that would result from the split contains fewer than `node.size` observation. Default is 0, which indicates that the node.size will be set automatically according to this formula: floor(sqrt(Number of training cases)).
`stop.prob`	if the proportion of the majority class in a tree node is greater than `stop.prob`, the node will become a leaf and no further split will be attempted on it.
`opt.solver`	a character string in the set 'enum', 'enum_c', 'gurobi', 'cplex', 'lpSolve', 'greedy' indicating the optimization solver to be used in the program. The choice of 'cplex' requires the package `cplexAPI`, 'gurobi' requires the package `gurobi`, 'lpSolve' requires the package `lpSolve` and 'enum_c' requires the .dll or .dylib file. The default is 'greedy' because it is fast and does not rely on other packages. The 'enum' algorithm is the implicit enumeration method which guarantees to find the optimal solution, typically faster than an optimization solver. It is a tradeoff between the greedy heuristic and the mathematical optimization methods.
`solver.timelimit`	the solver time limit in seconds. Currently only applicable to 'gurobi', 'enum' and 'enum_c' solvers.
`max.rules`	the maximum number of features allowed to enter an OR-clause split rule. A small max.rules reduces the search space and regulates model complexity. Default is 3.
`opt.model`	a character string in the set 'gini','error' indicating the optimization model to solve in the program. The default is 'gini'. The choice of 'error' is faster because the optimization model is smaller. The default is 'gini'.
`greedy.level`	a proportion value between 0 and 1, applicable only when opt.solver is 'greedy'. In the greedy forward selection process of split rules, a candidate rule is added to the OR-clause only if the split performance (gini reduction or accuracy) after the addition multiplied by greedy.level would still be greater than the split performance before the addition. A higher value of greedy.level tend to more aggressively produce multi-variable splits.
`import.external`	logical value indicating whether or not to try importing candidate split rules from other decision tree packages. Default is True.

`suppress.internal`	logical value indicating whether or not to suppress the feature binarization process that creates the pool of binary features. If it is set to True, then only the features imported from external methods (if import.external is True) will be used in the optimal rule selection model. Default is False.
`no.same.gender.children`	logical value indicating whether or not to suppress splits that would result in both children having the same majority class. Default is False.
`n0n1.cap`	a positive integer. It is applicable only when the opt.solver is 'hybrid' and the opt.model is 'gini'. When the bslearn function is called, if the product of the number of negative cases (n0) and the number of positive cases (n1) is greater than this number, 'enum' solver will be used; otherwise, gurobi solver will be used.
`verbose`	a logical value (TRUE or FALSE) indicating whether the solution details are to be printed on the screen.

Value

An object of class `bscontrol`.

Examples

```
bscontrol()  # display the default parameters
bsc <- bscontrol(stop.prob = 0.8, nseg.numeric = 10, verbose = TRUE)
bsc
```

`bslearn` *Find the Optimal Boolean Rule for Binary Classification*

Description

The function solves a mixed integer program (MIP) to either maximize the Gini reduction (opt.model = 'gini') or the number of misclassifications (opt.model = 'error'). The optimal rule serves as the split condition in the classification tree built by `bsnsing`.

Usage

```
bslearn(bx, y, control = bscontrol())
```

Arguments

`bx`	a data frame with binary (0 and 1) entries.
`y`	an integer vector with binary entries.
`control`	an object of class `bscontrol()`, specifying the algorithmic parameters. The list should contain the following attributes: *opt.model*, a character string in `'gini','error'` indicating the optimization model to solve, *opt.solver*, a character string in `'gurobi'`, `'cplex'`, `'lpSolve'`, `'enum'`, `'enum_c'`, `'greedy'` indicating the optimization method or solver to be used.

Value

a list containing the splitting solution.

Examples

```
## Not run:
x <- auto[, c('mpg', 'cylinders', 'displacement')]
y <- ifelse(auto$origin == 'USA', 1L, 0L)
# binarize x by y = 1
bx <- binarize(x, y, target = 1)
# learn the optimal Boolean rule
bssol <- bslearn(bx, y, bscontrol(opt.solver = 'enum'))
cat(paste("Optimal rule:" , bssol$rules, "\n"))

## End(Not run)
```

`bsnsing` *Learn a Classification Tree using Boolean Sensing*

Description

Depending on the arguments provided, either `bsnsing.default` or `bsnsing.formula` will be called.

Usage

```
bsnsing(x, ...)
```

Arguments

`x`	a data frame or a `formula` object.
`...`	arguments passed on to `bsnsing.default` or `bsnsing.formula`.

Value

an object of class `bsnsing` for a two-class problem or an object of class `mbsnsing` for a multi-class problem.

Examples

```
## Not run:
# Use the formula format
bs <- bsnsing(Class~., data = BreastCancer)
summary(bs)
# For multi-class classification
bs <- bsnsing(Species ~ ., data = iris)
summary(bs)
summary(bs[[1]])  # display the tree for the first class
summary(bs[[2]])  # display the tree for the second class
summary(bs[[3]])  # display the tree for the third class
predict(bs, type = 'class')  # the fitted class membership
predict(bs, type = 'prob')  # the fitted probabilities

# Use the (x, y) format, y must have two levels
y <- ifelse(iris$Species == 'setosa', 1L, 0L)
x <- iris[, c('Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width')]
bs <- bsnsing(x, y, verbose = TRUE)
summary(bs)

## End(Not run)
```

`bsnsing.default` *Learn a Classification Tree with Boolean Sensing*

Description

This is the default method for `bsnsing` and handles binary classification only. `bsnsing.formula`, which calls `bsnsing.default` as the basic tree builder, can handle multiclass classification problems. Missing values in numeric variables are imputed as the median of the non-missing ones, and missing values in factor variables are treated as a separate level named 'NA'.

Usage

```
## Default S3 method:
bsnsing(x, y, controls = bscontrol(), ...)
```

Arguments

`x`	a data frame containing independent variables. Columns can be of numeric, integer, factor and logical types. The column names must be proper identifiers (e.g., must start with a letter, cannot contain special characters and spaces, etc.).
`y`	a vector of the response variable. The response variable can be of an integer, numeric, logical or factor type, but must have only two unique values. Typical coding of a binary response variable is 0 (for negative case) and 1 (for positive cases).
`controls`	an object of class `bscontrol`.
`...`	further argument to be passed to bsnsing.default.

Value

an object of class `bsnsing`.

Examples

```
## Not run:
y <- ifelse(iris$Species == 'setosa', 1L, 0L)
x <- iris[, c('Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width')]
bs <- bsnsing(x, y, verbose = TRUE)
summary(bs)

## End(Not run)
```

`bsnsing.formula` *Learn a Classification Tree using Boolean Sensing*

Description

The program builds a binary classification tree for each unique value in the response variable. Each tree classifies a target value against all the other values (internally coded as 'all.other') in the response variable. If the response variable is a numeric type, the number of unique values must not exceed 30. There is no programmatic restriction on the number of unique levels for a factor response.

Usage

```
## S3 method for class 'formula'
bsnsing(formula, data, subset, na.action = stats::na.pass, ...)
```

Arguments

`formula`	an object of class "`formula`": a symbolic description of the model to be fitted.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `bsnsing.formula` is called.
`subset`	an optional vector specifying a subset (in terms of index numbers, not actual data) of observations to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain NAs. If `na.pass` is used, `bsnsing` will still apply an internal NA treatment logic, as follows: missing values in numeric variables will be replaced by the median of the non-missing values of the variable; missing values in factor variables will be treated as a spearate level named 'NA'.
`...`	additional arguments to be passed to the low level fitting functions, e.g., elements in the `bscontrol` object.

Value

an object of `bsnsing` for a two-class problem or an object of `mbsnsing` for a multi-class problem.

Examples

```
# Multi-class classification
## Not run:
bs <- bsnsing(Species ~ ., data = iris)
summary(bs)
summary(bs[[1]])  # display the tree for the first class
summary(bs[[2]])  # display the tree for the second class
summary(bs[[3]])  # display the tree for the third class

# Two-class classification
bs <- bsnsing(origin ~ mpg + displacement + horsepower + weight, data = auto, verbose = TRUE)
summary(bs)

## End(Not run)
```

`get_os` *Get the operating system type (windows, osx, linux).*

Description

This function is for internal use, to determine the file extension of the bslearn shared library. .dylib for oxs, .so for linux and .dll for windows.

Usage

```
get_os()
```

Value

a character string indicating the OS type

`GlaucomaMVF` *GlaucomaMVF*

Description

A test dataset

Usage

```
GlaucomaMVF
```

Format

A data frame with 170 rows and 67 variables:

Source

http://github.com/profyliu/bsnsing

iris — *iris*

Description

A test data set.

Usage

```
iris
```

Format

A data frame with 150 rows and 5 variables.

Source

http://github.com/profyliu/bsnsing

mbsnsing-class — *A class that contains multi-class classification model built by bsnsing. Can be used in summary and predict functions.*

Description

A class that contains multi-class classification model built by bsnsing. Can be used in summary and predict functions.

plot.bsnsing — *Generate latex code for plotting the bsnsing tree*

Description

If the file argument is supplied, this function will invoke the external programs latex, dvips and ps2pdf. If these programs are not available, only the latex code will be generated. If the file argument is left empty, the latex code will be written to the console screen. The latex code utilizes the following packages: pstricks, pst-node, pst-tree.

Usage

```
## S3 method for class 'bsnsing'
plot(
  object,
  file = "",
  class_labels = c(),
  class_colors = c("red", "green"),
  rule_font = c("footnotesize", "scriptsize", "tiny", "normalsize", "small"),
  rule_color = "blue",
  footnote = F,
  landscape = F,
 papersize = c("a0paper", "a1paper", "a2paper", "a3paper", "a4paper", "a5paper",
   "a6paper", "b0paper", "b1paper", "b2paper", "b3paper", "b4paper", "b5paper",
   "b6paper", "c0paper", "c1paper", "c2paper", "c3paper", "c4paper", "c5paper",
   "c6paper", "b0j", "b1j", "b2j", "b3j", "b4j", "b5j", "b6j", "ansiapaper",
   "ansibpaper", "ansicpaper", "ansidpaper", "ansiepaper", "letterpaper",
   "executivepaper", "legalpaper"),
  verbose = F,
  ...
)
```

Arguments

`object`	an object of class `bsnsing`.
`file`	a writable connection or a character string naming the file to write to. If not supplied, the output will be written to the console.
`class_labels`	a character vector of two elements for leaf node label (for 0 and 1). If empty, the labels will be read from the bsnsing object.
`class_colors`	a character vector of two elements for leaf node color (for 0 and 1)
`rule_font`	a string specifying the font size of the split rule at each non-leaf node
`rule_color`	a string specifying the color of the split rule and node, e.g., blue, gray, black, etc. For a list of all usable colors, see https://en.wikibooks.org/wiki/LaTeX/Colors
`footnote`	a boolean value indicating whether to add annotation text to the PDF page. The default is False.
`landscape`	a boolean value indicating if the PDF page should be in landscape layout. The default is False.
`papersize`	a string specifying the PDF paper size. The default is 'a0paper'.

Value

NA

Examples

```
# Suppose bs is a bsnsing object
## Not run:
```

```
plot(bs)
plot(bs, file = "/path/to/destination/filename.tex")

## End(Not run)
```

`plot.mbsnsing` *Generate latex code for plotting the bsnsing tree*

Description

Generate latex code for plotting the bsnsing tree

Usage

```
## S3 method for class 'mbsnsing'
plot(object)
```

Arguments

`object`	an object of class `mbsnsing`.

`predict.bsnsing` *Make Predictions with a Fitted* `bsnsing` *Model*

Description

Implements the generic `predict` function to make predictions on new data using a trained `bsnsing` model.

Usage

```
## S3 method for class 'bsnsing'
predict(object, newdata = NULL, type = c("prob", "class"), ...)
```

Arguments

`object`	a `bsnsing` model object.
`newdata`	a optional data frame in which to look for variables for prediction. If omitted, the fitted class or probability will be returned.
`type`	a character string indicating the type of prediction. *'prob'* predicts the probability of being a positive case (i.e., y = 1), and *'class'* predicts the class membership.
`...`	further arguments to predict.bsnsing.

Value

a vector containing the predicted values.

Examples

```
## Not run:
# Load data
n <- nrow(GlaucomaMVF)
train_index = sample(1:n, round(0.5*n))
test_index = setdiff(1:n, train_index)
# Fit a model using training set
bs <- bsnsing(Class ~ ., data = GlaucomaMVF, subset = train_index)
# Make predictions on the test set
pred <- predict(bs, GlaucomaMVF[test_index, ], type = 'class')
# Display the confusion matrix
table(pred, actual = GlaucomaMVF[test_index, 'Class'])

## End(Not run)
```

---

predict.mbsnsing *Make Predictions with a* bsnsing *Model*

Description

Make Predictions with a bsnsing Model

Usage

```
## S3 method for class 'mbsnsing'
predict(object, newdata = NULL, type = c("prob", "class"), ...)
```

Arguments

object	an object of class mbsnsing.
newdata	an optional data frame in which to look for variables for prediction. If omitted, the fitted class or probability will be returned.
type	a character string indicating the type of prediction. *'prob'* predicts the probability of being in each class, and *'class'* predicts the class membership.
...	further arguments to predict.mbsnsing.

Value

a data frame containing the predicted values.

Examples

```
## Not run:
n <- nrow(iris)
train_index <- sample(1:n, round(0.5*n))
test_index <- setdiff(1:n, train_index)
# Fit a model on the training set
bs <- bsnsing(Species ~ ., data = iris, subset = train_index)
# Make predictions on the test set
pred <- predict(bs, iris[test_index, ], type = 'class')
# Display the confusion matrix
table(pred, actual = iris[test_index, 'Species'])
# Predict the probabilities
predprob <- predict(bs, iris[test_index, ], type = 'prob')
head(predprob)

## End(Not run)
```

`print.bscontrol` *Print the Object of Class* `bscontrol`

Description

Print the Object of Class `bscontrol`

Usage

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

Arguments

`x`	an object of class `bscontrol`.
`...`	further arguments to the print function.

`print.bsnsing` *Print the Object of Class* `bsnsing`

Description

Print the Object of Class `bsnsing`

Usage

```
## S3 method for class 'bsnsing'
print(x, print.call = T, ...)
```

Arguments

`x`	an object of class `bsnsing`.
`print.call`	print out the function called, default TRUE
`...`	further arguments

`print.mbsnsing` *Print the Object of Class* `mbsnsing`

Description

Print the Object of Class `mbsnsing`

Usage

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

Arguments

`x`	an object of class `mbsnsing`.
`...`	further arguments.

`print.summary.bsnsing` *Print the Summary of* `bsnsing` *Model*

Description

Print the Summary of `bsnsing` Model

Usage

```
## S3 method for class 'summary.bsnsing'
print(x, print.call = T, ...)
```

Arguments

`x`	an object of class `summary.bsnsing`.
`print.call`	a logical value, print out the function called if TRUE.
`...`	further arguments.

print.summary.mbsnsing — *Print the summary of* `mbsnsing` *model fits*

Description

Print the summary of `mbsnsing` model fits

Usage

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

Arguments

`x`	an object of class `summary.mbsnsing`.
`...`	further arguments.

prt.bscontrol — *Print the Object of Class* `bscontrol`

Description

Print the Object of Class `bscontrol`

Usage

```
prt.bscontrol(control = bscontrol())
```

Arguments

`control`	an object of class `bscontrol`.

`prt.bsnsing` *Print the Object of Class* `bsnsing`

Description

Print the Object of Class `bsnsing`

Usage

```
prt.bsnsing(object, print.call = T, ...)
```

Arguments

`object`	an object of class `bsnsing`.
`print.call`	print out the function called, default TRUE
`...`	further arguments

`prt.mbsnsing` *Print the Object of Class* `mbsnsing`

Description

Print the Object of Class `mbsnsing`

Usage

```
prt.mbsnsing(object, ...)
```

Arguments

`object`	an object of class `mbsnsing`.
`...`	further arguments.

`prt.summary.bsnsing` *Print the Summary of `bsnsing` Model*

Description

Print the Summary of `bsnsing` Model

Usage

```
prt.summary.bsnsing(object, print.call = T, ...)
```

Arguments

`object`	an object of class `summary.bsnsing`.
`print.call`	a logical value, print out the function called if TRUE.
`...`	further arguments.

`prt.summary.mbsnsing` *Print the summary of `mbsnsing` model fits*

Description

Print the summary of `mbsnsing` model fits

Usage

```
prt.summary.mbsnsing(object, ...)
```

Arguments

`object`	an object of class `summary.mbsnsing`.
`...`	further arguments.

ROC_func *Plot the ROC curve and calculate the AUC*

Description

This is a general utility function, not part of the bsnsing functionality.

Usage

```
ROC_func(
  df,
  label_colnum,
  score_colnum,
  pos.label = "1",
  plot.ROC = F,
  add_on = F,
  color = "black",
  lty = 1
)
```

Arguments

`df`	a data frame which must contain at least these two columns: the prediction scores (numeric values, not necessarily be between 0 and 1) and the true class labels.
`label_colnum`	the column index of the scores column in df
`score_colnum`	the column index of the true class labels column in df
`pos.label`	a character string matching the positive class label used in the class labels column
`plot.ROC`	a logical value indicating whether the ROC curve should be plotted
`add_on`	a logical value indicating whether the ROC curve should be added to an existing plot
`color`	a character string specifying the color of the ROC curve in the plot
`lty`	line type used in the plot (1 solid, 2 dashed, etc.)

Value

the area under the curve (AUC) value

Examples

```
## Not run:
n <- nrow(BreastCancer)
trainset <- sample(1:n, 0.7*n)  # randomly sample 70\
testset <- setdiff(1:n, trainset)  # the remaining is for testing
# Build a tree to predict Class, using all default options
```

```
bs <- bsnsing(Class~., data = BreastCancer[trainset,])
summary(bs)  # display the tree structure
pred <- predict(bs, BreastCancer[testset,], type='class')
actual <- BreastCancer[testset, 'Class']
table(pred, actual)  # display the confusion matrix
# Plot the ROC curve and display the AUC
ROC_func(data.frame(predict(bs, BreastCancer[testset,]),
                    BreastCancer[testset,'Class']), 2, 1,
         pos.label = 'malignant', plot.ROC=TRUE)
# Plot the tree to PDF and generate the .tex file
plot(bs, file='../bsnsing_test/fig/BreastCancer.pdf')

## End(Not run)
```

show.bsnsing *Generate latex code for plotting the bsnsing tree*

Description

If the file argument is supplied, this function will invoke the external programs latex, dvips and ps2pdf. If these programs are not available, only the latex code will be generated. If the file argument is left empty, the latex code will be written to the console screen. The latex code utilizes the following packages: pstricks, pst-node, pst-tree.

Usage

```
show.bsnsing(
  object,
  file = "",
  class_labels = c(),
  class_colors = c("red", "green"),
  rule_font = c("footnotesize", "scriptsize", "tiny", "normalsize", "small"),
  rule_color = "blue",
  footnote = F,
  landscape = F,
 papersize = c("a0paper", "a1paper", "a2paper", "a3paper", "a4paper", "a5paper",
   "a6paper", "b0paper", "b1paper", "b2paper", "b3paper", "b4paper", "b5paper",
   "b6paper", "c0paper", "c1paper", "c2paper", "c3paper", "c4paper", "c5paper",
    "c6paper", "b0j", "b1j", "b2j", "b3j", "b4j", "b5j", "b6j", "ansiapaper",
    "ansibpaper", "ansicpaper", "ansidpaper", "ansiepaper", "letterpaper",
    "executivepaper", "legalpaper"),
  verbose = F,
  ...
)
```

Arguments

`object`	an object of class `bsnsing`.

`file`	a writable connection or a character string naming the file to write to. If not supplied, the output will be written to the console.
`class_labels`	a character vector of two elements for leaf node label (for 0 and 1). If empty, the labels will be read from the bsnsing object.
`class_colors`	a character vector of two elements for leaf node color (for 0 and 1)
`rule_font`	a string specifying the font size of the split rule at each non-leaf node
`rule_color`	a string specifying the color of the split rule and node, e.g., blue, gray, black, etc. For a list of all usable colors, see https://en.wikibooks.org/wiki/LaTeX/Colors
`footnote`	a boolean value indicating whether to add annotation text to the PDF page. The default is False.
`landscape`	a boolean value indicating if the PDF page should be in landscape layout. The default is False.
`papersize`	a string specifying the PDF paper size. The default is 'a0paper'.
`verbose`	a logical value, default is FALSE.
`...`	further parameters to the plot function.

Value

NA

Examples

```
# Suppose bs is a bsnsing object
## Not run:
plot(bs)
plot(bs, file = "/path/to/destination/filename.tex")

## End(Not run)
```

`show.mbsnsing` *Generate latex code for plotting the bsnsing tree*

Description

Generate latex code for plotting the bsnsing tree

Usage

```
show.mbsnsing(object, ...)
```

Arguments

`object`	an object of class `mbsnsing`.
`...`	further arguments to the plot function.

summary.bsnsing *Summarize the bsnsing Model Fits*

Description

Summarize the bsnsing Model Fits

Usage

```
## S3 method for class 'bsnsing'
summary(object = stop("no 'object' arg"), ...)
```

Arguments

`object`	an object of class `bsnsing`.
`...`	further arguments.

Value

a list of detailed information in the given `object`.

summary.mbsnsing *Summarize mbsnsing Model Fits*

Description

Summarize mbsnsing Model Fits

Usage

```
## S3 method for class 'mbsnsing'
summary(object = stop("no 'object' arg"), ...)
```

Arguments

`object`	an object of class `mbsnsing`.
`...`	further arguments.

Value

a list of detailed information in the given `object`.

Index