Package ‘traineR’

October 14, 2022

Type Package

Title Predictive (Classification and Regression) Models Homologator

Version 2.0.4

Depends R (>= 3.5)
Imports neuralnet (>= 1.44.2), rpart (>= 4.1-13), xgboost (>= 0.81.0.1), randomForest (>= 4.6-14), e1071 (>= 1.7-0.1), kknn (>= 1.3.1), dplyr (>= 0.8.0.1), MASS (>= 7.3-53), ada (>= 2.0-5), nnet (>= 7.3-12), stringr (>= 1.4.0), adabag, glmnet, ROCR, gbm, ggplot2


License GPL (>= 2)

Encoding UTF-8

URL https://www.promidat.com

RoxygenNote 7.2.1

NeedsCompilation no
Author Oldemar Rodriguez R. [aut, cre],
Andres Navarro D. [aut],
Ariel Arroyo S. [aut],
Diego Jimenez A. [ctb]

Maintainer Oldemar Rodriguez R. <oldemar.rodriguez@ucr.ac.cr>

Repository CRAN

Date/Publication 2022-08-22 22:50:02 UTC

R topics documented:

- boosting.importance.plot .................................................. 3
- categorical.predictive.power .............................................. 4
- confusion.matrix ............................................................. 5
- contr.dummy ................................................................. 6
- contr.metric ................................................................. 7
- contr.ordinal ............................................................... 7
- general.indexes ............................................................ 8
- numerical.predictive.power ................................................. 9
- plot.prdmt ................................................................. 10
- predict.ada.prdmt ......................................................... 10
- predict.adabag.prdmt ................................................... 11
- predict.bayes.prdmt ..................................................... 11
- predict.gbm.prdmt ....................................................... 12
- predict.glm.prdmt ....................................................... 13
- predict.glmnet.prdmt ................................................... 14
- predict.knn.prdmt ....................................................... 14
- predict.lda.prdmt ........................................................ 15
- predict.neuralnet.prdmt ................................................ 15
- predict.nnet.prdmt ....................................................... 16
- predict.qda.prdmt ......................................................... 16
- predict.randomForest.prdmt ............................................. 17
- predict.rpart.prdmt ..................................................... 18
- predict.svm.prdmt ...................................................... 18
- predict.xgb.Booster.prdmt ............................................. 19
- prediction.variable.balance ........................................... 20
- print.indexes.prdmt ..................................................... 21
- print.prediction.prdmt .................................................. 22
- print.prdmt ............................................................... 22
- ROC.area ................................................................. 23
- ROC.plot ................................................................. 23
- train.ada ................................................................. 24
- train.adabag ............................................................. 25
- train.bayes ............................................................. 27
- train.gbm ............................................................... 28
- train.glm ............................................................... 31
- train.glmnet ........................................................... 33
- train.knn ............................................................... 35
Description

Function that graphs the importance of the variables for a boosting model.

Usage

boosting.importance.plot(model, col = "steelblue")

Arguments

model fitted model object of class adabag.prmdt or boosting.
col the color of the chart bars.

Value

A ggplot object.

Note

With this function we can identify how important the variables are for the generation of a predictive model.

See Also

ggplot, train.adabag, boosting
Examples

```r
data <- iris
n <- nrow(data)

sam <- sample(1:n,n*0.75)
training <- data[sam,]
testing <- data[-sam,]

model <- train.adabag(formula = Species~.,data = training,minsplit = 2,
  maxdepth = 30, mfinal = 10)
boosting.importance.plot(model)
```

describe categorical.predictive.power

describe categorical.predictive.power

Description

Function that graphs the distribution of individuals and shows their category according to a categorical variable.

Usage

```r
categorical.predictive.power(
  data,
predict.variable,
  variable.to.compare,
ylab = "",
xlab = "",
main = paste("Variable Distribution", variable.to.compare, "according to",
  predict.variable),
col = NA
)
```

Arguments

- **data**: A data frame.
- **predict.variable**: Character type. The name of the variable to predict. This name must be part of the columns of the data frame.
- **variable.to.compare**: Character type. The name of the categorical variable to compare. This name must be part of the columns of the data frame.
- **ylab**: A character string that describes the y-axis on the graph.
- **xlab**: A character string that describes the x-axis on the graph.
**confusion.matrix**

- **main** Character type. The main title of the chart.
- **col** A vector that specifies the colors of the categories of the variable to predict.

**Value**

A ggplot object.

**Note**

With this function we can analyze the predictive power of a categorical variable.

**See Also**

`ggplot`

**Examples**

```r
  cars <- datasets::mtcars
  cars$cyl <- as.factor(cars$cyl)
  cars$vs <- as.factor(cars$vs)
  categorical.predictive.power(cars,"vs","cyl")
```

---

**Description**

create the confusion matrix.

**Usage**

```r
  confusion.matrix(newdata, prediction)
```

**Arguments**

- **newdata** matrix or data frame of test data.
- **prediction** a prmdt prediction object.

**Value**

A matrix with predicted and actual values.
Examples

```r
data("iris")
n <- seq_len(nrow(iris))
.sample <- sample(n, length(n) * 0.75)
data.train <- iris[.sample,]
data.test <- iris[-.sample,]

modelo.knn <- train.knn(Species~., data.train)
modelo.knn
prob <- predict(modelo.knn, data.test, type = "prob")
prob
prediccion <- predict(modelo.knn, data.test, type = "class")
prediccion
confusion.matrix(data.test, prediccion)
```

contr.dummy

```
contr.dummy
```

Description

Returns a matrix of contrasts for the `train.knn`.

Usage

```r
contr.dummy(n, contrasts = TRUE)
```

Arguments

- `n` A vector containing levels of a factor, or the number of levels.
- `contrasts` A logical value indicating whether contrasts should be computed.

Value

A matrix with n rows and n-1 columns for contr.ordinal, a matrix with n rows and n columns for contr.dummy and a vector of length n for contr.metric.
**Description**

Returns a matrix of contrasts for the `train.kknn`.

**Usage**

```r
contr.metric(n, contrasts = TRUE)
```

**Arguments**

- `n`: A vector containing levels of a factor, or the number of levels.
- `contrasts`: A logical value indicating whether contrasts should be computed.

**Value**

A matrix with n rows and n-1 columns for contr.ordinal, a matrix with n rows and n columns for contr.dummy and a vector of length n for contr.metric.

---

**Description**

Returns a matrix of contrasts for the `train.kknn`.

**Usage**

```r
contr.ordinal(n, contrasts = TRUE)
```

**Arguments**

- `n`: A vector containing levels of a factor, or the number of levels.
- `contrasts`: A logical value indicating whether contrasts should be computed.

**Value**

A matrix with n rows and n-1 columns for contr.ordinal, a matrix with n rows and n columns for contr.dummy and a vector of length n for contr.metric.
### Description

Calculates the confusion matrix, overall accuracy, overall error and the category accuracy for a classification problem and the Root Mean Square Error, Mean Absolute Error, Relative Error and Correlation for a regression problem.

### Usage

```r
general.indexes(newdata, prediction, mc = NULL)
```

### Arguments

- `newdata`: matrix or data frame of test data.
- `prediction`: a prmdt prediction object.
- `mc`: (optional) a matrix for calculating the indices. If `mc` is entered as parameter `newdata` and `prediction` are not necessary.

### Value

A list with the appropriate error and precision measurement. The class of this list is `indexes.prmdt`

### Examples

```r
# Classification
data("iris")

n <- seq_len(nrow(iris))
.sample <- sample(n, length(n) * 0.75)
data.train <- iris[, .sample]
data.test <- iris[-.sample]

modelo.knn <- train.knn(Species ~ ., data.train)
prediccion <- predict(modelo.knn, data.test, type = "class")
general.indexes(data.test, prediccion)

# Regression
len <- nrow(swiss)
samp1 <- sample(x = 1:len, size = len*0.20, replace = FALSE)
ttesting <- swiss[samp1]
ttraining <- swiss[-samp1]
model.knn <- train.knn(Infant.Mortality ~ ., ttraining)
prediccion <- predict(model.knn, ttesting)
prediccion
general.indexes(ttesting, prediccion)
```
**numerical.predictive.power**

**Description**

Function that graphs the density of individuals and shows their category according to a numerical variable.

**Usage**

```r
umerical.predictive.power(
data,        
predict.variable,       
variable.to.compare,      
ylab = "",              
xlab = "",              
main = paste("Variable Density", variable.to.compare, "according to", predict.variable), 
col = NA
)
```

**Arguments**

- `data`: A data frame.
- `predict.variable`: Character type. The name of the variable to predict. This name must be part of the columns of the data frame.
- `variable.to.compare`: Character type. The name of the numeric variable to compare. This name must be part of the columns of the data frame.
- `ylab`: A character string that describes the y-axis on the graph.
- `xlab`: A character string that describes the x-axis on the graph.
- `main`: Character type. The main title of the chart.
- `col`: A vector that specifies the colors of the categories of the variable to predict.

**Value**

A `ggplot` object.

**Note**

With this function we can analyze the predictive power of a numerical variable.

**See Also**

`ggplot`
Examples

```r
numerical.predictive.power(iris,"Species","Sepal.Length")
```

plot.prmdt  
Plotting prmdt models

Description

Plotting prmdt models

Usage

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

Arguments

- `x`: A prmdt models
- `...`: optional arguments to print o format method

Value

- a plot of a model.

Description

Return prediction for a ada model.

Usage

```r
# S3 method for class 'ada.prmdt'
predict(object, newdata, type = "class", n.iter = NULL, ...)
```

Arguments

- `object`: a ada model object for which prediction is desired.
- `newdata`: an optional data frame in which to look for variables with which to predict.
- `type`: type of prediction 'prob' or 'class' (default).
- `n.iter`: number of iterations to consider for the prediction. By default this is iter from the ada call (n.iter< iter).
- `...`: additional arguments affecting the predictions produced.
Value

a vector or matrix of predictions for ada model.

Description

Return prediction for a **boosting** model.

Usage

```r
## S3 method for class 'adabag.prmdt'
predict(object, newdata, type = "class", ...)
```

Arguments

- **object**: a **boosting** model object for which prediction is desired.
- **newdata**: an optional data frame in which to look for variables with which to predict.
- **type**: type of prediction 'prob' or 'class' (default).
- **...**: additional arguments affecting the predictions produced.

Value

a vector or matrix of predictions adabag model.

Description

Return prediction for a **naiveBayes** model.

Usage

```r
## S3 method for class 'bayes.prmdt'
predict(object, newdata, type = "class", threshold = 0.001, eps = 0, ...)
```

Arguments

- **object**: a **naiveBayes** model object for which prediction is desired.
- **newdata**: an optional data frame in which to look for variables with which to predict.
- **type**: type of prediction 'prob' or 'class' (default).
- **threshold**: Value replacing cells with 0 probabilities.
- **eps**: double for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by theshold).
- **...**: additional arguments affecting the predictions produced.
Value

a vector or matrix of predictions for bayes model.

Description

Return prediction for a gbm model.

Usage

## S3 method for class 'gbm.prmdt'
predict(
  object,
  newdata,
  type = "class",
  n.trees = NULL,
  single.tree = FALSE,
  ...
)

Arguments

object a gbm model object for which prediction is desired.
newdata an optional data frame in which to look for variables with which to predict.
type type of prediction 'prob' or 'class' (default).
n.trees Number of trees used in the prediction. n.trees may be a vector in which case predictions are returned for each iteration specified
single.tree If single.tree=TRUE then predict.gbm returns only the predictions from tree(s) n.trees.
... additional arguments affecting the predictions produced.

Value

a vector or matrix of predictions gbm model.
predict.glm.prmdt

Description

Return prediction for a glm model.

Usage

## S3 method for class 'glm.prmdt'
predict(
  object,
  newdata,
  type = "class",
  se.fit = FALSE,
  dispersion = NULL,
  terms = NULL,
  na.action = na.pass,
  ...
)

Arguments

object a glm model object for which prediction is desired.
newdata an optional data frame in which to look for variables with which to predict.
type type of prediction 'prob' or 'class' (default).
se.fit logical switch indicating if standard errors are required.
dispersion the dispersion of the GLM fit to be assumed in computing the standard errors. If omitted, that returned by summary applied to the object is used.
terms with type = "terms" by default all terms are returned. A character vector specifies which terms are to be returned.
na.action function determining what should be done with missing values in newdata. The default is to predict NA.
... additional arguments affecting the predictions produced.

Value

a vector or matrix of predictions for glm model.
### predict.glmnet.prmdt

**Description**

Return prediction for a `glmnet` model.

**Usage**

```r
## S3 method for class 'glmnet.prmdt'
predict(object, newdata, type = "class", s = NULL, ...)
```

**Arguments**

- `object`: a `glmnet` model object for which prediction is desired.
- `newdata`: an optional data frame in which to look for variables with which to predict.
- `type`: type of prediction 'prob' or 'class' (default).
- `s`: a `cv.glmnet` object (optional).
- `...`: additional arguments affecting the predictions produced.

### predict.knn.prmdt

**Description**

Return prediction for a `train.kknn` model.

**Usage**

```r
## S3 method for class 'knn.prmdt'
predict(object, newdata, type = "class", ...)
```

**Arguments**

- `object`: a `train.kknn` model object for which prediction is desired.
- `newdata`: an optional data frame in which to look for variables with which to predict.
- `type`: type of prediction 'prob' or 'class' (default).
- `...`: additional arguments affecting the predictions produced.

**Value**

- a vector or matrix of predictions for knn model.
**predict.lda.prmdt**

### Description
Return prediction for a **lda** model.

### Usage
```r
## S3 method for class 'lda.prmdt'
predict(object, newdata, type = "class", ...)
```

### Arguments
- **object**: a **lda** model object for which prediction is desired.
- **newdata**: an optional data frame in which to look for variables with which to predict.
- **type**: type of prediction 'prob' or 'class' (default).
- **...**: additional arguments affecting the predictions produced.

### Value
a vector or matrix of predictions for lda model.

**predict.neuralnet.prmdt**

### Description
Return prediction for a **neuralnet** model.

### Usage
```r
## S3 method for class 'neuralnet.prmdt'
predict(object, newdata, type = "class", ...)
```

### Arguments
- **object**: a **neuralnet** model object for which prediction is desired.
- **newdata**: an optional data frame in which to look for variables with which to predict.
- **type**: type of prediction 'prob' or 'class' (default).
- **...**: additional arguments affecting the predictions produced.

### Value
a vector or matrix of predictions for neuralnet.
Description

Return prediction for a \texttt{nnet} model.

Usage

\begin{verbatim}
## S3 method for class 'nnet.prmdt'
predict(object, newdata, type = "class", ...)
\end{verbatim}

Arguments

- \texttt{object}: a \texttt{nnet} model object for which prediction is desired.
- \texttt{newdata}: an optional data frame in which to look for variables with which to predict.
- \texttt{type}: type of prediction 'prob' or 'class' (default).
- \texttt{...}: additional arguments affecting the predictions produced.

Value

a vector or matrix of predictions for \texttt{nnet} model.

Description

Return prediction for a \texttt{qda} model.

Usage

\begin{verbatim}
## S3 method for class 'qda.prmdt'
predict(object, newdata, type = "class", ...)
\end{verbatim}

Arguments

- \texttt{object}: a \texttt{qda} model object for which prediction is desired.
- \texttt{newdata}: an optional data frame in which to look for variables with which to predict.
- \texttt{type}: type of prediction 'prob' or 'class' (default).
- \texttt{...}: additional arguments affecting the predictions produced.

Value

a vector or matrix of predictions for \texttt{qda} model.
Description

Return prediction for a randomForest model.

Usage

```r
## S3 method for class 'randomForest.prmdt'
predict(
  object,
  newdata,  # an optional data frame in which to look for variables with which to predict.
  type = "class",  # type of prediction 'prob' or 'class' (default).
  norm.votes = TRUE,  # Should the vote counts be normalized (i.e., expressed as fractions)? Ignored if object$type is regression.
  predict.all = FALSE,  # Should the predictions of all trees be kept?
  proximity = FALSE,  # Should proximity measures be computed? An error is issued if object$type is regression.
  nodes = FALSE,  # Should the terminal node indicators (an n by ntree matrix) be return? If so, it is in the "nodes" attribute of the returned object.
  cutoff,  # (Classification only) A vector of length equal to number of classes. The 'winning' class for an observation is the one with the maximum ratio of proportion of votes to cutoff. Default is taken from the forest$cutoff component of object (i.e., the setting used when running randomForest).
  ...  # additional arguments affecting the predictions produced.
)
```

Arguments

- **object**: a randomForest model object for which prediction is desired.
- **newdata**: an optional data frame in which to look for variables with which to predict.
- **type**: type of prediction 'prob' or 'class' (default).
- **norm.votes**: Should the vote counts be normalized (i.e., expressed as fractions)? Ignored if object$type is regression.
- **predict.all**: Should the predictions of all trees be kept?
- **proximity**: Should proximity measures be computed? An error is issued if object$type is regression.
- **nodes**: Should the terminal node indicators (an n by ntree matrix) be return? If so, it is in the "nodes" attribute of the returned object.
- **cutoff**: (Classification only) A vector of length equal to number of classes. The ‘winning’ class for an observation is the one with the maximum ratio of proportion of votes to cutoff. Default is taken from the forest$cutoff component of object (i.e., the setting used when running randomForest).
- **...**: additional arguments affecting the predictions produced.

Value

A vector or matrix of predictions for randomforest model.
predict.rpart.prmdt

Description
Return prediction for a \texttt{rpart} model.

Usage

## S3 method for class 'rpart.prmdt'
predict(object, newdata, type = "class", na.action = na.pass, ...)

Arguments

- \texttt{object}: a \texttt{rpart} model object for which prediction is desired.
- \texttt{newdata}: an optional data frame in which to look for variables with which to predict.
- \texttt{type}: type of prediction 'prob' or 'class' (default).
- \texttt{na.action}: a function to determine what should be done with missing values in \texttt{newdata}. The default is to pass them down the tree using surrogates in the way selected when the model was built. Other possibilities are \texttt{na.omit} and \texttt{na.fail}.
- ...: additional arguments affecting the predictions produced.

Value
a vector or matrix of predictions for \texttt{rpart} model.

predict.svm.prmdt

Description
Return prediction for a \texttt{svm} model.

Usage

## S3 method for class 'svm.prmdt'
predict(  
  object,  
  newdata,  
  type = "class",  
  decision.values = FALSE,  
  ...,
  na.action = na.omit
)


Arguments

- **object**: a svm model object for which prediction is desired.
- **newdata**: an optional data frame in which to look for variables with which to predict.
- **type**: type of prediction 'prob' or 'class' (default).
- **decision.values**: Logical controlling whether the decision values of all binary classifiers computed in multiclass classification shall be computed and returned.
- **...**: additional arguments affecting the predictions produced.
- **na.action**: A function to specify the action to be taken if ‘NA’s are found. The default action is na.omit, which leads to rejection of cases with missing values on any required variable. An alternative is na.fail, which causes an error if NA cases are found. (NOTE: If given, this argument must be named.)

Value

a vector or matrix of predictions for svm model.

Description

Return prediction for a xgb.train model.

Usage

```r
## S3 method for class 'xgb.Booster.prmdt'
predict(
  object,
  newdata,
  type = "class",
  missing = NA,
  outputmargin = FALSE,
  ntreelimit = NULL,
  predleaf = FALSE,
  predcontrib = FALSE,
  approxcontrib = FALSE,
  predinteraction = FALSE,
  reshape = FALSE,
  ...
)
```
**Arguments**

- `object`: a `xgb.train` model object for which prediction is desired.
- `newdata`: an optional data frame in which to look for variables with which to predict.
- `type`: type of prediction 'prob' or 'class' (default).
- `missing`: Missing is only used when input is dense matrix. Pick a float value that represents missing values in data (e.g., sometimes 0 or some other extreme value is used).
- `outputmargin`: whether the prediction should be returned in the form of original untransformed sum of predictions from boosting iterations' results. E.g., setting `outputmargin=TRUE` for logistic regression would result in predictions for log-odds instead of probabilities.
- `ntreelimit`: Deprecated, use `iterationrange` instead.
- `predleaf`: whether predict leaf index.
- `predcontrib`: whether to return feature contributions to individual predictions (see Details).
- `approxcontrib`: whether to use a fast approximation for feature contributions (see Details).
- `predinteraction`: whether to return contributions of feature interactions to individual predictions (see Details).
- `reshape`: whether to reshape the vector of predictions to a matrix form when there are several prediction outputs per case. This option has no effect when either of `predleaf`, `predcontrib`, or `predinteraction` flags is `TRUE`.
- `...`: additional arguments affecting the predictions produced.

**Value**

A vector or matrix of predictions for xgb model.

**Description**

Function that graphs the balance of the different categories of a column of a data frame.

**Usage**

```r
prediction.variable.balance(
  data, 
  predict.variable, 
  ylab = "Number of individuals", 
  xlab = "", 
  main = paste("Variable Distribution", predict.variable), 
  col = NA 
)
```
Arguments

- **data**: A data frame.
- **predict.variable**: Character type. The name of the variable to predict. This name must be part of the columns of the data frame.
- **ylab**: A character string that describes the y-axis on the graph.
- **xlab**: A character string that describes the x-axis on the graph.
- **main**: Character type. The main title of the chart.
- **col**: A vector that specifies the colors of the categories represented by bars within the chart.

Value

A ggplot object.

Note

With this function we can identify if the data is balanced or not, according to the variable to be predicted.

See Also

- [ggplot](#)

Examples

```r
prediction.variable.balance(iris,"Species")
```

---

Description

Printing prmdt index object

Usage

```r
## S3 method for class 'indexes.prmdt'
print(x, ...)
```

Arguments

- **x**: A prmdt index object
- **...**: optional arguments to print o format method
Value

a print of the results of a prediction model.

---

print.prediction.prmdt

*Printing prmdt prediction object*

---

Description

Printing prmdt prediction object

Usage

```r
## S3 method for class 'prediction.prmdt'
print(x, ...)
```

Arguments

- `x` A prmdt prediction object
- `...` optional arguments to print o format method

Value

a print prediction of a model.

---

print.prmdt

*Printing prmdt models*

---

Description

Printing prmdt models

Usage

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

Arguments

- `x` A prmdt models
- `...` optional arguments to print o format method

Value

a print information of a model.
**ROC.area**

**Description**

Function that calculates the area of the ROC curve of a prediction with only 2 categories.

**Usage**

ROC.area(prediction, real)

**Arguments**

- **prediction**: A vector of real numbers representing the prediction score of a category.
- **real**: A vector with the real categories of the individuals in the prediction.

**Value**

The value of the area (numeric).

**See Also**

prediction and performance

**Examples**

```r
iris2 <- dplyr::filter(iris,(Species == "setosa") | (Species == "virginica"))
iris2$Species <- factor(iris2$Species,levels = c("setosa","virginica"))
sam <- sample(1:100,20)
ttesting <- iris2[sam,]
ttraining <- iris2[-sam,]
model <- train.rpart(Species~.,ttraining)
prediction.prob <- predict(model,ttesting, type = "prob")
ROC.area(prediction.prob$s$prediction[,2],ttesting$Species)
```

---

**ROC.plot**

**Description**

Function that plots the ROC curve of a prediction with only 2 categories.

**Usage**

ROC.plot(prediction, real, .add = FALSE, color = "red")
train.ada

Arguments

- **prediction**: A vector of real numbers representing the prediction score of a category.
- **real**: A vector with the real categories of the individuals in the prediction.
- **.add**: A logical value that indicates if it should be added to an existing graph
- **color**: Color of the ROC curve in the graph

Value

A plot object.

See Also

`prediction` and `performance`

Examples

```r
iris2 <- dplyr::filter(iris,(Species == "setosa") | (Species == "virginica"))
iris$Species <- factor(iris$Species,levels = c("setosa","virginica"))
sam <- sample(1:100,20)
ttesting <- iris2[sam,]
ttraining <- iris2[-sam,]
model <- train.rpart(Species~.,ttraining)
prediction.prob <- predict(model,ttesting, type = "prob")
ROC.plot(prediction.prob$prediction[,2],ttesting$Species)
```

train.ada

Description

Provides a wrapping function for the `ada`.

Usage

`train.ada(formula, data, ... , subset, na.action = na.rpart)`

Arguments

- **formula**: a symbolic description of the model to be fit.
- **data**: an optional data frame containing the variables in the model.
- **...**: arguments passed to `rpart.control`. For stumps, use `rpart.control(maxdepth=1,cp=-1,msplit=0,xval=0)`. maxdepth controls the depth of trees, and cp controls the complexity of trees. The priors should also be fixed through the `parms` argument as discussed in the second reference.
- **subset**: an optional vector specifying a subset of observations to be used in the fitting process.
- **na.action**: a function that indicates how to process 'NA' values. Default=na.rpart.
**train.adabag**

**Value**

A object ada.prmdt with additional information to the model that allows to homogenize the results.

**Note**

the parameter information was taken from the original function `ada`.

**See Also**

The internal function is from package `ada`.

**Examples**

```r
data("Puromycin")

n <- seq_len(nrow(Puromycin))
.sample <- sample(n, length(n) * 0.75)
data.train <- Puromycin[.sample,]
data.test <- Puromycin[-.sample,]

modelo.ada <- train.ada(state~., data.train)
modelo.ada
prob <- predict(modelo.ada, data.test , type = "prob")
prob
prediccion <- predict(modelo.ada, data.test , type = "class")
prediccion
```

**Description**

Provides a wrapping function for the `boosting`.

**Usage**

```r
train.adabag(
  formula,
  data,
  boos = TRUE,
  mfinal = 100,
  coeflearn = "Breiman",
  msplit = 20,
  maxdepth = 30,
  ...
)
```
Arguments

formula a symbolic description of the model to be fit.
data an optional data frame containing the variables in the model.
boos if TRUE (by default), a bootstrap sample of the training set is drawn using the weights for each observation on that iteration. If FALSE, every observation is used with its weights.
mfinal an integer, the number of iterations for which boosting is run or the number of trees to use. Defaults to mfinal=100 iterations.
coeflearn if 'Breiman' (by default), alpha=1/2ln((1-err)/err) is used. If 'Freund' alpha=ln((1-err)/err) is used. In both cases the AdaBoost.M1 algorithm is used and alpha is the weight updating coefficient. On the other hand, if coeflearn is 'Zhu' the SAMME algorithm is implemented with alpha=ln((1-err)/err)+ ln(nclasses-1).
minsplit the minimum number of observations that must exist in a node in order for a split to be attempted.
maxdepth Set the maximum depth of any node of the final tree, with the root node counted as depth 0. Values greater than 30 rpart will give nonsense results on 32-bit machines.
... arguments passed to rpart.control or adabag::boosting. For stumps, use rpart.control(maxdepth=1,cp=-1,minsplit=0,xval=0). maxdepth controls the depth of trees, and cp controls the complexity of trees.

Value

A object adabag.prmdt with additional information to the model that allows to homogenize the results.

Note

The parameter information was taken from the original function boosting and rpart.control.

See Also

The internal function is from package boosting.

Examples

data <- iris
n <- nrow(data)
sam <- sample(1:n,n*0.75)
training <- data[sam,]
testing <- data[-sam,]

model <- train.adabag(formula = Species~.,data = training,minsplit = 2,
                       maxdepth = 30, mfinal = 10)
model
predict <- predict(object = model,testing,type = "class")
Description

Provides a wrapping function for the naiveBayes.

Usage

train.bayes(formula, data, laplace = 0, ..., subset, na.action = na.pass)

Arguments

formula A formula of the form class ~ x1 + x2 + .... Interactions are not allowed.
data Either a data frame of predictors (categorical and/or numeric) or a contingency table.
laplace positive double controlling Laplace smoothing. The default (0) disables Laplace smoothing.
... Currently not used.
subset For data given in a data frame, an index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
na.action A function to specify the action to be taken if NAs are found. The default action is not to count them for the computation of the probability factors. An alternative is na.omit, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)

Value

A object bayes.prmdt with additional information to the model that allows to homogenize the results.

Note

the parameter information was taken from the original function naiveBayes.

See Also

The internal function is from package naiveBayes.
Examples

# Classification
data("iris")

n <- seq_len(nrow(iris))
.sample <- sample(n, length(n) * 0.75)
data.train <- iris[, .sample]
data.test <- iris[-.sample]

modelo.bayes <- train.bayes(Species ~ ., data.train)
modelo.bayes
prob <- predict(modelo.bayes, data.test, type = "prob")
prob
prediccion <- predict(modelo.bayes, data.test, type = "class")
prediccion

# Regression
len <- nrow(swiss)
sampl <- sample(x = 1:len, size = len*0.20, replace = FALSE)
ttesting <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.bayes <- train.bayes(Infant.Mortality~., ttraining)
prediction <- predict(model.bayes, ttesting)
prediction

Description

Provides a wrapping function for the \texttt{gbm}.

Usage

\begin{verbatim}
train.gbm(
  formula,
  data,
  distribution = "bernoulli",
  weights,
  var.monotone = NULL,
  n.trees = 100,
  interaction.depth = 1,
  n.minobsinnode = 10,
  shrinkage = 0.001,
  bag.fraction = 0.5,
  train.fraction = 1,
\end{verbatim}
Arguments

- **formula**: a symbolic description of the model to be fit.
- **data**: an optional data frame containing the variables in the model.
- **distribution**: Either a character string specifying the name of the distribution to use or a list with a component name specifying the distribution and any additional parameters needed.
- **weights**: an optional vector of weights to be used in the fitting process. Must be positive but do not need to be normalized.
- **var.monotone**: an optional vector, the same length as the number of predictors, indicating which variables have a monotone increasing (+1), decreasing (-1), or arbitrary (0) relationship with the outcome.
- **n.trees**: Integer specifying the total number of trees to fit. This is equivalent to the number of iterations and the number of basis functions in the additive expansion. Default is 100.
- **interaction.depth**: Integer specifying the maximum depth of each tree (i.e., the highest level of variable interactions allowed). A value of 1 implies an additive model, a value of 2 implies a model with up to 2-way interactions, etc. Default is 1.
- **n.minobsinnode**: Integer specifying the minimum number of observations in the terminal nodes of the trees. Note that this is the actual number of observations, not the total weight.
- **shrinkage**: a shrinkage parameter applied to each tree in the expansion. Also known as the learning rate or step-size reduction; 0.001 to 0.1 usually work, but a smaller learning rate typically requires more trees. Default is 0.1.
- **bag.fraction**: the fraction of the training set observations randomly selected to propose the next tree in the expansion. This introduces randomnesses into the model fit.
- **train.fraction**: The first train.fraction * nrow(data) observations are used to fit the gbm and the remainder are used for computing out-of-sample estimates of the loss function.
- **cv.folds**: Number of cross-validation folds to perform. If cv.folds>1 then gbm, in addition to the usual fit, will perform a cross-validation, calculate an estimate of generalization error returned in cv.error.
- **keep.data**: a logical variable indicating whether to keep the data and an index of the data stored with the object. Keeping the data and index makes subsequent calls to gbm.more faster at the cost of storing an extra copy of the dataset.
- **verbose**: Logical indicating whether or not to print out progress and performance indicators (TRUE). If this option is left unspecified for gbm.more, then it uses verbose from object. Default is FALSE.
class.stratify.cv

Logical indicating whether or not the cross-validation should be stratified by class.

n.cores

The number of CPU cores to use. The cross-validation loop will attempt to send different CV folds off to different cores. If n.cores is not specified by the user, it is guessed using the detectCores function in the parallel package.

Value

A object gbm.prmdt with additional information to the model that allows to homogenize the results.

Note

The parameter information was taken from the original function gbm.

See Also

The internal function is from package gbm.

Examples

# Classification
data <- iris
n <- nrow(data)
sam <- sample(1:n, n*0.75)
training <- data[sam,]
testing <- data[-sam,]
model <- train.gbm(formula = Species ~ ., data = training)
model
predict <- predict(object = model, testing)
predict

# Regression
len <- nrow(swiss)
sampl <- sample(x = 1:len, size = len*0.10, replace = FALSE)
ttesting <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.gbm <- train.gbm(Infant.Mortality~., ttraining, distribution = "gaussian")
prediction <- predict(model.gbm, ttesting)
prediction
Description

Provides a wrapping function for the \texttt{glm}

Usage

\begin{verbatim}
train.glm(
  formula, data,
  family = binomial, weights, subset, na.action, start = NULL,
  etastart, mustart, offset, control = list(...), model = TRUE,
  method = "glm.fit", x = FALSE, y = TRUE,
  singular.ok = TRUE, contrasts = NULL,
  ...
)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{formula} an object of class "formula" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under ‘Details’.
  \item \texttt{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 glm is called.
  \item \texttt{family} a description of the error distribution and link function to be used in the model. For \texttt{glm} this can be a character string naming a family function, a family function or the result of a call to a family function. For \texttt{glm.fit} only the third option is supported. (See family for details of family functions.)
  \item \texttt{weights} an optional vector of ‘prior weights’ to be used in the fitting process. Should be \texttt{NULL} or a numeric vector.
\end{itemize}
subset

an optional vector specifying a subset of observations to be used in the fitting process.

na.action

a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The ‘factory-fresh’ default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.

start

starting values for the parameters in the linear predictor.

etastart

starting values for the linear predictor.

mustart

starting values for the vector of means.

offset

this can be used to specify an a priori known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See model.offset.

control

a list of parameters for controlling the fitting process. For glm.fit this is passed to glm.control.

model

a logical value indicating whether model frame should be included as a component of the returned value.

method

the method to be used in fitting the model. The default method "glm.fit" uses iteratively reweighted least squares (IWLS): the alternative "model.frame" returns the model frame and does no fitting. User-supplied fitting functions can be supplied either as a function or a character string naming a function, with a function which takes the same arguments as glm.fit. If specified as a character string it is looked up from within the stats namespace.

x, y

For glm: logical values indicating whether the response vector and model matrix used in the fitting process should be returned as components of the returned value. For glm.fit: x is a design matrix of dimension n * p, and y is a vector of observations of length n.

singular.ok

logical; if FALSE a singular fit is an error.

contrasts

an optional list. See the contrasts.arg of model.matrix.default.

...

For glm: arguments to be used to form the default control argument if it is not supplied directly. For weights: further arguments passed to or from other methods.

Value

A object glm prm dt with additional information to the model that allows to homogenize the results.

See Also

The internal function is from package glm.

The internal function is from package glm.
Examples

# Classification
data("Puromycin")

n <- seq_len(nrow(Puromycin))
.sample <- sample(n, length(n) * 0.65)
data.train <- Puromycin[, .sample]
data.test <- Puromycin[-.sample]

modelo.glm <- train.glm(state ~ ., data.train)
modelo.glm
prob <- predict(modelo.glm, data.test, type = "prob")
prob
prediccion <- predict(modelo.glm, data.test, type = "class")
prediccion

# Regression
len <- nrow(swiss)
sampl <- sample(x = 1:len, size = len*0.20, replace = FALSE)
testing <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.glm <- train.glm(Infant.Mortality ~ ., ttraining, family = "gaussian")
prediction <- predict(model.glm, testing)
prediction

Description

Provides a wrapping function for the glmnet.

Usage

train.glmnet(
  formula,
  data,
  standardize = TRUE,
  alpha = 1,
  family = "multinomial",
  cv = TRUE,
  ...)

Arguments

formula A formula of the form groups ~ x1 + x2 + ... That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.

data An optional data frame, list or environment from which variables specified in formula are preferentially to be taken.

standardize Logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is standardize=TRUE. If variables are in the same units already, you might not wish to standardize. See details below for y standardization with family="gaussian".

alpha The elasticnet mixing parameter. alpha=1 is the lasso penalty, and alpha=0 the ridge penalty.

family Either a character string representing one of the built-in families, or else a glm() family object. For more information, see Details section below or the documentation for response type (above).

cv True or False. Perform cross-validation to find the best value of the penalty parameter lambda and save this value in the model. This value could be used in predict() function.

... Arguments passed to or from other methods.

Value

A object glmnet.prmdt with additional information to the model that allows to homogenize the results.

Note

The parameter information was taken from the original function glmnet.

See Also

The internal function is from package glmnet.

Examples

# Classification
len <- nrow(iris)
sampl <- sample(x = 1:len,size = len*0.20,replace = FALSE)
ttesting <- iris[sampl,]
ttraining <- iris[-sampl,]
model.glmnet <- train.glmnet(Species~.,ttraining)
prediction <- predict(model.glmnet,ttesting)
prediction

# Regression
len <- nrow(swiss)
sampl <- sample(x = 1:len,size = len*0.20,replace = FALSE)
ttesting <- swiss[sampl,]
train.knn

```r
training <- swiss[-sampl,]
model.glmnet <- train.glmnet(Infant.Mortality~.,training, family = "gaussian")
prediction <- predict(model.glmnet, ttesting)
prediction
```

---

**train.knn**

**Description**

Provides a wrapping function for the `train.kknn`.

**Usage**

```r
train.knn(
  formula, 
  data, 
  kmax = 11, 
  ks = NULL, 
  distance = 2, 
  kernel = "optimal", 
  ykernel = NULL, 
  scale = TRUE, 
  contrasts = c(unordered = "contr.dummy", ordered = "contr.ordinal"), 
  ...
)
```

**Arguments**

- `formula` A formula object.
- `data` Matrix or data frame.
- `kmax` Maximum number of k, if ks is not specified.
- `ks` A vector specifying values of k. If not null, this takes precedence over kmax.
- `distance` Parameter of Minkowski distance.
- `kernel` Kernel to use. Possible choices are "rectangular" (which is standard unweighted knn), "triangular", "epanechnikov" (or beta(2,2)), "biweight" (or beta(3,3)), "triweight" (or beta(4,4)), "cos", "inv", "gaussian" and "optimal".
- `ykernel` Window width of an y-kernel, especially for prediction of ordinal classes.
- `scale` logical, scale variable to have equal sd.
- `contrasts` A vector containing the 'unordered' and 'ordered' contrasts to use.
- `...` Further arguments passed to or from other methods.

**Value**

A object knn.prmdt with additional information to the model that allows to homogenize the results.
Note

the parameter information was taken from the original function \texttt{train.kknn}.

See Also

The internal function is from package \texttt{train.kknn}.

Examples

```r
# Classification
data("iris")

n <- seq_len(nrow(iris))
.sample <- sample(n, length(n) * 0.75)
data.train <- iris[.sample,]
data.test <- iris[-.sample,]

modelo.knn <- train.knn(Species~., data.train)
modelo.knn

prob <- predict(modelo.knn, data.test, type = "prob")
prob

prediccion <- predict(modelo.knn, data.test, type = "class")
prediccion

# Regression
len <- nrow(swiss)
sampl <- sample(x = 1:len, size = len*0.20, replace = FALSE)
ttesting <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.knn <- train.knn(Infant.Mortality~., ttraining)
prediction <- predict(model.knn, ttesting)
prediction
```

Description

Provides a wrapping function for the \texttt{lda}.

Usage

\texttt{train.lda(formula, data, \ldots, subset, na.action)}
**train.neuralnet**

**Arguments**

- **formula**
  A formula of the form `groups ~ x1 + x2 + ...` That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.

- **data**
  An optional data frame, list or environment from which variables specified in formula are preferentially to be taken.

- **...**
  Arguments passed to or from other methods.

- **subset**
  An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)

- **na.action**
  Function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is na.omit, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)

**Value**

A object lda.prmdt with additional information to the model that allows to homogenize the results.

**Note**

The parameter information was taken from the original function `lda`.

**See Also**

The internal function is from package `lda`.

**Examples**

```r
len <- nrow(iris)
sampl <- sample(x = 1:len, size = len*0.20, replace = FALSE)
ttesting <- iris[sampl,]
ttraining <- iris[-sampl,]
model.lda <- train.lda(Species~, ttraining)
model.lda
prediction <- predict(model.lda, ttesting)
prediction
```

---

**Description**

Provides a wrapping function for the `neuralnet`.
Usage

```r
train.neuralnet(
  formula,
  data,
  hidden = 1,
  threshold = 0.01,
  stepmax = 1e+05,
  rep = 1,
  startweights = NULL,
  learningrate.limit = NULL,
  learningrate.factor = list(minus = 0.5, plus = 1.2),
  learningrate = NULL,
  lifesign = "none",
  lifesign.step = 1000,
  algorithm = "rprop+",
  err.fct = "sse",
  act.fct = "logistic",
  linear.output = TRUE,
  exclude = NULL,
  constant.weights = NULL,
  likelihood = FALSE
)
```

Arguments

- **formula**: a symbolic description of the model to be fitted.
- **data**: a data frame containing the variables specified in formula.
- **hidden**: a vector of integers specifying the number of hidden neurons (vertices) in each layer.
- **threshold**: a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.
- **stepmax**: the maximum steps for the training of the neural network. Reaching this maximum leads to a stop of the neural network's training process.
- **rep**: the number of repetitions for the neural network's training.
- **startweights**: a vector containing starting values for the weights. Set to NULL for random initialization.
- **learningrate.limit**: a vector or a list containing the lowest and highest limit for the learning rate. Used only for RPROP and GRPROP.
- **learningrate.factor**: a vector or a list containing the multiplication factors for the upper and lower learning rate. Used only for RPROP and GRPROP.
- **learningrate**: a numeric value specifying the learning rate used by traditional backpropagation. Used only for traditional backpropagation.
- **lifesign**: a string specifying how much the function will print during the calculation of the neural network. 'none', 'minimal' or 'full'.
lifesign.step an integer specifying the stepsize to print the minimal threshold in full lifesign mode.

algorithm a string containing the algorithm type to calculate the neural network. The following types are possible: 'backprop', 'rprop+', 'rprop-', 'sag', or 'slr'. 'backprop' refers to backpropagation, 'rprop+' and 'rprop-' refer to the resilient backpropagation with and without weight backtracking, while 'sag' and 'slr' induce the usage of the modified globally convergent algorithm (grprop). See Details for more information.

err.fct a differentiable function that is used for the calculation of the error. Alternatively, the strings 'sse' and 'ce' which stand for the sum of squared errors and the cross-entropy can be used.

act.fct a differentiable function that is used for smoothing the result of the cross product of the covariate or neurons and the weights. Additionally the strings, 'logistic' and 'tanh' are possible for the logistic function and tangent hyperbolicus.

linear.output logical. If act.fct should not be applied to the output neurons set linear output to TRUE, otherwise to FALSE.

exclude a vector or a matrix specifying the weights, that are excluded from the calculation. If given as a vector, the exact positions of the weights must be known. A matrix with n-rows and 3 columns will exclude n weights, where the first column stands for the layer, the second column for the input neuron and the third column for the output neuron of the weight.

constant.weights a vector specifying the values of the weights that are excluded from the training process and treated as fix.

likelihood logical. If the error function is equal to the negative log-likelihood function, the information criteria AIC and BIC will be calculated. Furthermore the usage of confidence.interval is meaningful.

Value

A object neuralnet.prdt with additional information to the model that allows to homogenize the results.

Note

the parameter information was taken from the original function neuralnet.

See Also

The internal function is from package neuralnet.
train.nnet

Description

Provides a wrapping function for the nnet.

Usage

train.nnet(formula, data, weights, ..., subset, na.action, contrasts = NULL)

Arguments

- **formula**: A formula of the form class ~ x1 + x2 + ...
- **data**: Data frame from which variables specified in formula are preferentially to be taken.
- **weights**: (case) weights for each example – if missing defaults to 1.
- **subset**: An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
- **na.action**: A function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is na.omit, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
- **contrasts**: a list of contrasts to be used for some or all of the factors appearing as variables in the model formula.

Value

A object nnet.prmdt with additional information to the model that allows to homogenize the results.

Note

the parameter information was taken from the original function nnet.

See Also

The internal function is from package nnet.

Examples

```r
# Classification
data("iris")

n <- seq_len(nrow(iris))
.sample <- sample(n, length(n) * 0.75)
```
train.qda

```r
data.train <- iris[, sample,]
data.test <- iris[-sample,]

modelo.nn <- train.nnet(Species~., data.train, size = 20)
modelo.nn
prob <- predict(modelo.nn, data.test, type = "prob")
prob
prediccion <- predict(modelo.nn, data.test, type = "class")
prediccion

# Regression
len <- nrow(swiss)
sampl <- sample(x = 1:len, size = len*0.20, replace = FALSE)
ttesting <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.knn <- train.nnet(Infant.Mortality~., ttraining, size = 20)
prediction <- predict(model.knn, ttesting)
prediction
```

train.qda  

### Description

Provides a wrapping function for the `qda`

### Usage

```r
train.qda(formula, data, ..., subset, na.action)
```

### Arguments

- **formula**: A formula of the form `groups ~ x1 + x2 + ...` That is, the response is the grouping factor and the right hand side specifies the (non-factor) discriminators.
- **data**: An optional data frame, list or environment from which variables specified in formula are preferentially to be taken.
- **...**: Arguments passed to or from other methods.
- **subset**: An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
- **na.action**: Function to specify the action to be taken if NAs are found. The default action is for the procedure to fail. An alternative is `na.omit`, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)

### Value

A object `qda.prmdt` with additional information to the model that allows to homogenize the results.
train.randomForest

Note

The parameter information was taken from the original function qda.

See Also

The internal function is from package qda.

Examples

```r
len <- nrow(iris)
sampl <- sample(x = 1:len, size = len*0.20, replace = FALSE)
ttesting <- iris[sampl,]
ttraining <- iris[-sampl,]
model.qda <- train.qda(Species~., ttraining)
model.qda
prediction <- predict(model.qda, ttesting)
prediction
```

train.randomForest

Description

Provides a wrapping function for the randomForest.

Usage

```r
train.randomForest(formula, data, ..., subset, na.action = na.fail)
```

Arguments

- **formula**: a formula describing the model to be fitted (for the print method, an randomForest object).
- **data**: an optional data frame containing the variables in the model. By default the variables are taken from the environment which randomForest is called from.
- **...**: optional parameters to be passed to the low level function randomForest.default.
- **subset**: an index vector indicating which rows should be used. (NOTE: If given, this argument must be named.)
- **na.action**: A function to specify the action to be taken if NAs are found. (NOTE: If given, this argument must be named.)

Value

A object randomForest.pmdt with additional information to the model that allows to homogenize the results.
train.rpart

Note

the parameter information was taken from the original function randomForest.

See Also

The internal function is from package randomForest.

Examples

# Classification
data("iris")

n <- seq_len(nrow(iris))
.sample <- sample(n, length(n) * 0.75)
data.train <- iris[, .sample,]
data.test <- iris[-.sample,]

modelo.rf <- train.randomForest(Species~, data.train)
modelo.rf
prob <- predict(modelo.rf, data.test, type = "prob")
prob
prediccion <- predict(modelo.rf, data.test, type = "class")
prediccion

# Regression
len <- nrow(swiss)
sampl <- sample(x = 1:len, size = len*0.20, replace = FALSE)
ttesting <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.rf <- train.randomForest(Infant.Mortality~, ttraining)
prediction <- predict(modelo.rf, ttesting)
prediction
na.action = na.rpart,
method,
model = TRUE,
x = FALSE,
y = TRUE,
parms,
control,
cost,
...
)

Arguments

formula  a formula, with a response but no interaction terms. If this a a data frame, that is taken as the model frame.
data  an optional data frame in which to interpret the variables named in the formula.
weights  optional case weights.
subset  optional expression saying that only a subset of the rows of the data should be used in the fit.
na.action  the default action deletes all observations for which y is missing, but keeps those in which one or more predictors are missing.
method  one of "anova", "poisson", "class" or "exp". If method is missing then the routine tries to make an intelligent guess. If y is a survival object, then method = "exp" is assumed, if y has 2 columns then method = "poisson" is assumed, if y is a factor then method = "class" is assumed, otherwise method = "anova" is assumed. It is wisest to specify the method directly, especially as more criteria may added to the function in future. Alternatively, method can be a list of functions named init, split and eval. Examples are given in the file ‘tests/usersplits.R’ in the sources, and in the vignettes ‘User Written Split Functions’.
model  if logical: keep a copy of the model frame in the result? If the input value for model is a model frame (likely from an earlier call to the rpart function), then this frame is used rather than constructing new data.
x  keep a copy of the x matrix in the result.
y  keep a copy of the dependent variable in the result. If missing and model is supplied this defaults to FALSE.
parms  optional parameters for the splitting function. Anova splitting has no parameters. Poisson splitting has a single parameter, the coefficient of variation of the prior distribution on the rates. The default value is 1. Exponential splitting has the same parameter as Poisson. For classification splitting, the list can contain any of: the vector of prior probabilities (component prior), the loss matrix (component loss) or the splitting index (component split). The priors must be positive and sum to 1. The loss matrix must have zeros on the diagonal and positive off-diagonal elements. The splitting index can be gini or information. The default priors are proportional to the data counts, the losses default to 1, and the split defaults to gini.
control  a list of options that control details of the rpart algorithm. See rpart.control.
train.rpart

cost  a vector of non-negative costs, one for each variable in the model. Defaults to one for all variables. These are scalings to be applied when considering splits, so the improvement on splitting on a variable is divided by its cost in deciding which split to choose.

...  arguments to rpart.control may also be specified in the call to rpart. They are checked against the list of valid arguments.

Value

A object rpart.pmdt with additional information to the model that allows to homogenize the results.

Note

the parameter information was taken from the original function rpart.

See Also

The internal function is from package rpart.

Examples

# Classification
data("iris")

n <- seq_len(nrow(iris))
.sample <- sample(n, length(n) * 0.75)
data.train <- iris[,sample,]
data.test <- iris[-.sample,]

modelo.rpart <- train.rpart(Species~., data.train)
modelo.rpart
prob <- predict(modelo.rpart, data.test, type = "prob")
prob
prediccion <- predict(modelo.rpart, data.test, type = "class")
prediccion

# Regression
len <- nrow(swiss)
sampl <- sample(x = 1:len, size = len*0.20,replace = FALSE)
ttesting <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.rpart <- train.rpart(Infant.Mortality~.,ttraining)
prediction <- predict(model.rpart,ttesting)
prediction
train.svm

**Description**

Provides a wrapping function for the `svm`.

**Usage**

```r
train.svm(formula, data, ..., subset, na.action = na.omit, scale = TRUE)
```

**Arguments**

- **formula**: a symbolic description of the model to be fit.
- **data**: an optional data frame containing the variables in the model. By default the variables are taken from the environment which `svm` is called from.
- **...**: additional parameters for the low level fitting function svm.default
- **subset**: An index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
- **na.action**: A function to specify the action to be taken if NAs are found. The default action is na.omit, which leads to rejection of cases with missing values on any required variable. An alternative is na.fail, which causes an error if NA cases are found. (NOTE: If given, this argument must be named.)
- **scale**: A logical vector indicating the variables to be scaled. If scale is of length 1, the value is recycled as many times as needed. Per default, data are scaled internally (both x and y variables) to zero mean and unit variance. The center and scale values are returned and used for later predictions.

**Value**

A object svm.prmdt with additional information to the model that allows to homogenize the results.

**Note**

the parameter information was taken from the original function `svm`.

**See Also**

The internal function is from package `svm`.

**Examples**

```r
# Classification
data("iris")

n <- seq_len(nrow(iris))
```
.sample <- sample(n, length(n) * 0.75)
data.train <- iris[.sample,]
data.test <- iris[-.sample,]

modelo.svm <- train.svm(Species~, data.train)
modelo.svm
prob <- predict(modelo.svm, data.test, type = "prob")
prob
prediccion <- predict(modelo.svm, data.test, type = "class")
prediccion

# Regression
len <- nrow(swiss)
sampl <- sample(x = 1:len, size = len*0.20, replace = FALSE)
ttesting <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.svm <- train.svm(Infant.Mortality~., ttraining)
prediction <- predict(model.svm, ttesting)
prediction

---

**Description**

Provides a wrapping function for the `xgb.train`.

**Usage**

```r
train.xgboost(
  formula,
  data,
  nrounds,
  watchlist = list(),
  obj = NULL,
  feval = NULL,
  verbose = 1,
  print_every_n = 1L,
  early_stopping_rounds = NULL,
  maximize = NULL,
  save_period = NULL,
  save_name = "xgboost.model",
  xgb_model = NULL,
  callbacks = list(),
  eval_metric = "mlogloss",
  extra_params = NULL,
  booster = "gbtree",
  objective = NULL,
)```
eta = 0.3,
gamma = 0,
max_depth = 6,
min_child_weight = 1,
subsample = 1,
colsample_bytree = 1,
...)

Arguments

formula a symbolic description of the model to be fit.
data training dataset. xgb.train accepts only an xgb.DMatrix as the input. xgboost, in addition, also accepts matrix, dgCMatrix, or name of a local data file.
nrounds max number of boosting iterations.
watchlist named list of xgb.DMatrix datasets to use for evaluating model performance. Metrics specified in either eval_metric or feval will be computed for each of these datasets during each boosting iteration, and stored in the end as a field named evaluation_log in the resulting object. When either verbose>=1 or cb.print.evaluation callback is engaged, the performance results are continuously printed out during the training. E.g., specifying watchlist=list(validation1=mat1, validation2=mat2) allows to track the performance of each round’s model on mat1 and mat2.
obj customized objective function. Returns gradient and second order gradient with given prediction and dtrain.
feval custimized evaluation function. Returns list(metric='metric-name', value='metric-value') with given prediction and dtrain.
verbose If 0, xgboost will stay silent. If 1, it will print information about performance. If 2, some additional information will be printed out. Note that setting verbose > 0 automatically engages the cb.print.evaluation(period=1) callback function.
print_every_n Print each n-th iteration evaluation messages when verbose>0. Default is 1 which means all messages are printed. This parameter is passed to the cb.print.evaluation callback.
early_stopping_rounds If NULL, the early stopping function is not triggered. If set to an integer k, training with a validation set will stop if the performance doesn’t improve for k rounds. Setting this parameter engages the cb.early.stop callback.
maximize If feval and early_stopping_rounds are set, then this parameter must be set as well. When it is TRUE, it means the larger the evaluation score the better. This parameter is passed to the cb.early.stop callback.
save_period when it is non-NULL, model is saved to disk after every save_period rounds. 0 means save at the end. The saving is handled by the cb.save.model callback.
save_name the name or path for periodically saved model file.
xgb_model a previously built model to continue the training from. Could be either an object of class xgb.Booster, or its raw data, or the name of a file with a previously saved model.
callbacks  a list of callback functions to perform various task during boosting. See callbacks. Some of the callbacks are automatically created depending on the parameters’ values. User can provide either existing or their own callback methods in order to customize the training process.

eval_metric  eval_metric evaluation metrics for validation data. Users can pass a self-defined function to it. Default: metric will be assigned according to objective (rmse for regression, and error for classification, mean average precision for ranking). List is provided in detail section.

extra_params  the list of parameters. The complete list of parameters is available at http://xgboost.readthedocs.io/en/latest/parameter.html.

booster  booster which booster to use, can be gbtree or gblinear. Default: gbtree.

objective  objective specify the learning task and the corresponding learning objective, users can pass a self-defined function to it. The default objective options are below: + reg:linear linear regression (Default). + reg:logistic logistic regression. + binary:logistic logistic regression for binary classification. Output probability. + binary:logitraw logistic regression for binary classification, output score before logistic transformation. + num_class set the number of classes. To use only with multiclass objectives. + multi:softmax set xgboost to do multiclass classification using the softmax objective. Class is represented by a number and should be from 0 to num_class - 1. + multi:softprob same as softmax, but prediction outputs a vector of ndata * nclass elements, which can be further reshaped to ndata, nclass matrix. The result contains predicted probabilities of each data point belonging to each class. + rank:pairwise set xgboost to do ranking task by minimizing the pairwise loss.

eta  eta control the learning rate: scale the contribution of each tree by a factor of 0 < eta < 1 when it is added to the current approximation. Used to prevent overfitting by making the boosting process more conservative. Lower value for eta implies larger value for nrounds: low eta value means model more robust to overfitting but slower to compute. Default: 0.3

gamma  gamma minimum loss reduction required to make a further partition on a leaf node of the tree. the larger, the more conservative the algorithm will be. gamma minimum loss reduction required to make a further partition on a leaf node of the tree. the larger, the more conservative the algorithm will be.

max_depth  max_depth maximum depth of a tree. Default: 6

min_child_weight  min_child_weight minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression mode, this simply corresponds to minimum number of instances needed to be in each node. The larger, the more conservative the algorithm will be. Default: 1

subsample  subsample subsample ratio of the training instance. Setting it to 0.5 means that xgboost randomly collected half of the data instances to grow trees and this will prevent overfitting. It makes computation shorter (because less data to analyse). It is advised to use this parameter with eta and increase nrounds. Default: 1

colsample_bytree  colsample_bytree subsample ratio of columns when constructing each tree. Default: 1
... other parameters to pass to params.

Value

A object xgb.Booster.pmtdt with additional information to the model that allows to homogenize the results.

Note

the parameter information was taken from the original function xgb.train.

See Also

The internal function is from package xgb.train.

Examples

# Classification
data("iris")

n <- seq_len(nrow(iris))
.sample <- sample(n, length(n) * 0.75)
data.train <- iris[.sample,]
data.test <- iris[-.sample,]

modelo.xg <- train.xgboost(Species~., data.train, nrounds = 79, maximize = FALSE)
modelo.xg
prob <- predict(modelo.xg, data.test, type = "prob")
prob
prediccion <- predict(modelo.xg, data.test, type = "class")
prediccion

# Regression
len <- nrow(swiss)
samp1 <- sample(x = 1:len, size = len*0.20, replace = FALSE)
ttesting <- swiss[samp1,]
ttraining <- swiss[-samp1,]
model.xgb <- train.xgboost(Infant.Mortality~., ttraining, nrounds = 79, maximize = FALSE)
prediction <- predict(modelo.xg, ttesting)
prediction
Description


Details

```
Package: traineR
Type: Package
Version: 2.0.4
Date: 2022-08-22
License: GPL (>=2)
```

Author(s)

Maintainer: Oldemar Rodriguez Rojas <oldemar.rodriguez@ucr.ac.cr>

- Oldemar Rodriguez Rojas <oldemar.rodriguez@ucr.ac.cr>
- Andres Navarro D
- Ariel Arroyo S
- Diego Jiménez

---

**Description**

Plotting prmdt ada models

**Usage**

```
varplot(x, ...)
```
Arguments

- `x`    A ada prmdt model
- `...` optional arguments to print o format method

Value

A plot of the importance of variables.
Index

* package
  traineR, 50

  ada, 10, 24, 25
boosting, 3, 11, 25, 26
boosting.importance.plot, 3
categorical.predictive.power, 4
confusion.matrix, 5
contr.dummy, 6
contr.metric, 7
contr.ordinal, 7
cv.glmnet, 14
gbm, 12, 28, 30
general.indexes, 8
ggplot, 3, 5, 9, 21
glm, 13, 31, 32
glmnet, 14, 33, 34
lda, 15, 36, 37
naiveBayes, 11, 27
neuralnet, 15, 37, 39
nnet, 16, 40
numerical.predictive.power, 9
performance, 23, 24
plot.prmdt, 10
predict.ada.prmdt, 10
predict.adabag.prmdt, 11
predict.bayes.prmdt, 11
predict.gbm.prmdt, 12
predict.glm.prmdt, 13
predict.glmnet.prmdt, 14
predict.knn.prmdt, 14
predict.lda.prmdt, 15
predict.neuralnet.prmdt, 15
predict.nnet.prmdt, 16
predict.qda.prmdt, 16
predict.randomForest.prmdt, 17
predict.rpart.prmdt, 18
predict.svm.prmdt, 18
predict.xgb.Booster.prmdt, 19
prediction, 23, 24
prediction.variable.balance, 20
print.indexes.prmdt, 21
print.prediction.prmdt, 22
print.prmdt, 22
qda, 16, 41, 42
randomForest, 17, 42, 43
ROC.area, 23
ROC.plot, 23
rpart, 18, 43, 45
rpart.control, 26, 44, 45
svm, 18, 19, 46
train.ada, 24
train.adabag, 3, 25
train.bayes, 27
train.gbm, 28
train.glm, 31
train.glmnet, 33
train.kknn, 6, 7, 14, 35, 36
train.knn, 35
train.lda, 36
train.neuralnet, 37
train.nnet, 40
train.qda, 41
train.randomForest, 42
train.rpart, 43
train.svm, 46
train.xgboost, 47
traineR, 50
varplot, 51
xgb.train, 19, 20, 47, 50