Package 'traineR'

April 30, 2021

Type Package

Title Predictive Models Homologator

Version 1.6.1

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), dummies (>= 1.5.6), stringr (>= 1.4.0), adabag, glmnet, ROCR, ggplot2, scales, glue, grDevices

Suggests knitr, rmarkdown, rpart.plot

Description Methods to unify the different ways of creating predictive models and their different predictive formats. It includes
methods such as K-Nearest Neighbors, Decision Trees, ADA Boosting, Extreme Gradient Boosting, Random Forest, Neural Networks, Deep Learning,
Support Vector Machines, Bayesian Methods, Linear Discriminant Analysis and Quadratic Discriminant Analysis, Logistic Regression, Penalized Logistic Regression.

License GPL (>= 2)

Encoding UTF-8

URL https://www.promidat.com

RoxygenNote 7.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Oldemar Rodriguez R. [aut, cre],
Andres Navarro D. [ctb, prg],
Ariel Arroyo S. [ctb, prg]

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

Repository CRAN

Date/Publication 2021-04-30 09:50:02 UTC

R topics documented:

`boosting.importance.plot`

boosting.importance.plot

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

```
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)
```

`categorical.predictive.power` *categorical.predictive.power*

Description

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

Usage

```
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.
`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

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

`confusion.matrix` *confusion.matrix*

Description

create the confusion matrix.

Usage

```
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

```
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)
```

---

general.indexes *general.indexes*

Description

Calculates the confusion matrix, overall accuracy, overall error and the category accuracy

Usage

```
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 confusion matrix, overall accuracy, overall error and the category accuracy. The class of this list is indexes.prmdt

Examples

```
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
general.indexes(data.test, prediccion)
```

`numerical.predictive.power` *numerical.predictive.power*

Description

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

Usage

```
numerical.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

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

---

prediction.variable.balance

prediction.variable.balance

Description

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

Usage

```
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

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

`ROC.area` *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

```
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$prediction[,2],ttesting$Species)
```

`ROC.plot` *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")
```

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

```
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.plot(prediction.prob$prediction[,2],ttesting$Species)
```

`train.ada` *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,minsplit=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.

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

```
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
confusion.matrix(data.test, prediccion)
```

`train.adabag` *train.adabag*

Description

Provides a wrapping function for the `boosting`.

Usage

```
train.adabag(
  formula,
  data,
  boos = TRUE,
  mfinal = 100,
  coeflearn = "Breiman",
  minsplit = 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)
predict <- predict(object = model,testing,type = "class")
MC <- confusion.matrix(testing,predict)
general.indexes(mc = MC)
```

---

train.bayes *train.bayes*

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

```
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
confusion.matrix(data.test, prediccion)
```

train.glm *train.glm*

Description

Provides a wrapping function for the glm

Usage

```
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,
  ...
)
```

Arguments

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'.
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.
family	a description of the error distribution and link function to be used in the model. For 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 glm.fit only the third option is supported. (See family for details of family functions.)
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or a numeric vector.

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.prmdt 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

```
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
confusion.matrix(data.test, prediccion)
```

`train.glmnet` *train.glmnet*

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

```
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
general.indexes(ttesting,prediction)
```

---

train.knn *train.knn*

Description

Provides a wrapping function for the train.kknn.

Usage

```
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 train.kknn.

See Also

The internal function is from package train.kknn.

Examples

```
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)
```

`train.lda` *train.lda*

Description

Provides a wrapping function for the `lda`.

Usage

```
train.lda(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 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

```
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)
prediction <- predict(model.lda,ttesting)
prediction
general.indexes(ttesting,prediction)
```

---

train.neuralnet *train.neuralnet*

Description

Provides a wrapping function for the neuralnet.

Usage

```
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 meaningfull.

Value

A object neuralnet.prmdt 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`.

Examples

```
data("iris")

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

modelo.neuralnet <- train.neuralnet(Species~., data.train,hidden = c(10, 14, 13),
                                    linear.output = FALSE, threshold = 0.01, stepmax = 1e+06)
modelo.neuralnet
prob <- predict(modelo.neuralnet, data.test, type = "prob")
prob
prediccion <- predict(modelo.neuralnet, data.test, type = "class")
prediccion
confusion.matrix(data.test, prediccion)
```

---

train.nnet *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.
...	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	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

```
data("iris")

n <- seq_len(nrow(iris))
.sample <- sample(n, length(n) * 0.75)
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
confusion.matrix(data.test, prediccion)
```

train.qda *train.qda*

Description

Provides a wrapping function for the qda.

Usage

```
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.

Note

The parameter information was taken from the original function qda.

See Also

The internal function is from package qda.

Examples

```
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)
prediction <- predict(model.qda,ttesting)
prediction
```

```
general.indexes(ttesting,prediction)
```

`train.randomForest` *train.randomForest*

Description

Provides a wrapping function for the `randomForest`.

Usage

```
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.prmdt with additional information to the model that allows to homogenize the results.

Note

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

See Also

The internal function is from package `randomForest`.

Examples

```
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
confusion.matrix(data.test, prediccion)
```

`train.rpart` *train.rpart*

Description

Provides a wrapping function for the `rpart`.

Usage

```
train.rpart(
  formula,
  data,
  weights,
  subset,
  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.
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.prmdt 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

```
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
confusion.matrix(data.test, prediccion)
```

`train.svm` *train.svm*

Description

Provides a wrapping function for the `svm`.

Usage

```
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

```
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
confusion.matrix(data.test, prediccion)
```

---

train.xgboost *train.xgboost*

Description

Provides a wrapping function for the xgb.train.

Usage

```
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/lates

`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.prmdt 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

```
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
confusion.matrix(data.test, prediccion)
```

varplot *Plotting prmdt ada models*

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