

# Package ‘traineR’

May 23, 2025

**Type** Package

**Title** Predictive (Classification and Regression) Models Homologator

**Version** 2.2.2

**Depends** R (>= 4.1)

**Imports** neuralnet (>= 1.44.2), rpart (>= 4.1-13), xgboost (>= 0.81.0.1), randomForest (>= 4.6-14), e1071 (>= 1.7-0.1), kknm (>= 1.4.1), 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

**Description** Methods to unify the different ways of creating predictive models and their different predictive formats for classification and regression. It includes methods such as K-Nearest Neighbors Schliep, K. P. (2004) <[doi:10.5282/ubm/epub.1769](https://doi.org/10.5282/ubm/epub.1769)>, Decision Trees Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. Stone (2017) <[doi:10.1201/9781315139470](https://doi.org/10.1201/9781315139470)>, ADA Boosting Esteban Alfaro, Matias Gamez, Noelia García (2013) <[doi:10.18637/jss.v054.i02](https://doi.org/10.18637/jss.v054.i02)>, Extreme Gradient Boosting Chen & Guestrin (2016) <[doi:10.1145/2939672.2939785](https://doi.org/10.1145/2939672.2939785)>, Random Forest Breiman (2001) <[doi:10.1023/A:1010933404324](https://doi.org/10.1023/A:1010933404324)>, Neural Networks Venables, W. N., & Ripley, B. D. (2002) <ISBN:0-387-95457-0>, Support Vector Machines Bennett, K. P. & Campbell, C. (2000) <[doi:10.1145/380995.380999](https://doi.org/10.1145/380995.380999)>, Bayesian Methods Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (1995) <[doi:10.1201/9780429258411](https://doi.org/10.1201/9780429258411)>, Linear Discriminant Analysis Venables, W. N., & Ripley, B. D. (2002) <ISBN:0-387-95457-0>, Quadratic Discriminant Analysis Venables, W. N., & Ripley, B. D. (2002) <ISBN:0-387-95457-0>, Logistic Regression Dobson, A. J., & Barnett, A. G. (2018) <[doi:10.1201/9781315182780](https://doi.org/10.1201/9781315182780)> and Penalized Logistic Regression Friedman, J. H., Hastie, T., & Tibshirani, R. (2010) <[doi:10.18637/jss.v033.i01](https://doi.org/10.18637/jss.v033.i01)>.

**License** GPL (>= 2)

**Encoding** UTF-8

**URL** <https://promidat.website/>

**BugReports** <https://github.com/PROMiDAT/traineR/issues>

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Oldemar Rodriguez R. [aut, cre],  
 Andres Navarro D. [aut],  
 Ariel Arroyo S. [aut],  
 Diego Jimenez A. [aut]

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

**Repository** CRAN

**Date/Publication** 2025-05-23 19:40:02 UTC

## Contents

categorical.predictive.power . . . . .	3
confusion.matrix . . . . .	4
contr.dummy . . . . .	5
contr.metric . . . . .	5
contr.ordinal . . . . .	6
general.indexes . . . . .	6
importance.plot . . . . .	7
numerical.predictive.power . . . . .	8
plot.prmdt . . . . .	9
predict.ada.prmdt . . . . .	10
predict.adabag.prmdt . . . . .	10
predict.bayes.prmdt . . . . .	11
predict.gbm.prmdt . . . . .	12
predict.glm.prmdt . . . . .	12
predict.glmnet.prmdt . . . . .	13
predict.knn.prmdt . . . . .	14
predict.lda.prmdt . . . . .	14
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.variable.balance . . . . .	20
print.indexes.prmdt . . . . .	21
print.prediction.prmdt . . . . .	22
print.prmdt . . . . .	22
ROC.area . . . . .	23
ROC.plot . . . . .	23
scaler . . . . .	24
train.ada . . . . .	25
train.adabag . . . . .	26
train.bayes . . . . .	27
train.gbm . . . . .	29
train.glm . . . . .	31
train.glmnet . . . . .	33

train.knn . . . . .	35
train.lda . . . . .	37
train.neuralnet . . . . .	38
train.nnet . . . . .	40
train.qda . . . . .	41
train.randomForest . . . . .	42
train.rpart . . . . .	44
train.svm . . . . .	46
train.xgboost . . . . .	47
traineR . . . . .	51
varplot . . . . .	52

## Index 53

---

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	<i>confusion.matrix</i>
------------------	-------------------------

---

**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)

```

---

 contr.dummy

*contr.dummy*


---

### Description

Returns a matrix of contrasts for the [train.kknn](#).

### Usage

```
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](#).

---

 contr.metric

*contr.metric*


---

### Description

Returns a matrix of contrasts for the [train.kknn](#).

### Usage

```
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`.

---

<code>contr.ordinal</code>	<i>contr.ordinal</i>
----------------------------	----------------------

---

**Description**

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

**Usage**

```
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`.

---

<code>general.indexes</code>	<i>general.indexes</i>
------------------------------	------------------------

---

**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**

```
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.prmtd`

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

---

importance.plot	<i>importance.plot</i>
-----------------	------------------------

---

**Description**

Function that graphs the importance of the variables.

**Usage**

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

**Arguments**

model	fitted model object.
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,minsplitt = 2,
  maxdepth = 30, mfinal = 10)
importance.plot(model)
```

---

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")
```

---

plot.prmtd

*Plotting prmdt models*

---

**Description**

Plotting prmdt models

**Usage**

```
## 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.

---

predict.ada.prmtdt      *predict.ada.prmtdt*

---

**Description**

Return prediction for a [ada](#) model.

**Usage**

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

**Arguments**

object	a <a href="#">ada</a> 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.

---

predict.adabag.prmtdt      *predict.adabag.prmtdt*

---

**Description**

Return prediction for a [boosting](#) model.

**Usage**

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

**Arguments**

object	a <a href="#">boosting</a> 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.

---

predict.bayes.prdmt    *predict.bayes.prdmt*

---

**Description**

Return prediction for a [naiveBayes](#) model.

**Usage**

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

**Arguments**

object	a <a href="#">naiveBayes</a> 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.

---

predict.gbm.prmtd      *predict.gbm.prmtd*

---

### Description

Return prediction for a [gbm](#) model.

### Usage

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

### Arguments

object	a <a href="#">gbm</a> 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.prmtd      *predict.glm.prmtd*

---

### Description

Return prediction for a [glm](#) model.

**Usage**

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

**Arguments**

object	a <a href="#">glm</a> 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.prmtd    *predict.glmnet.prmtd*

---

**Description**

Return prediction for a [glmnet](#) model.

**Usage**

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

**Arguments**

object	a <a href="#">glmnet</a> 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 <a href="#">cv.glmnet</a> object (optional).
...	additional arguments affecting the predictions produced.

---

`predict.knn.prdmt`      *predict.knn.prdmt*

---

**Description**

Return prediction for a [train.kknn](#) model.

**Usage**

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

**Arguments**

object	a <a href="#">train.kknn</a> 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.prdmt`      *predict.lda.prdmt*

---

**Description**

Return prediction for a [lda](#) model.

**Usage**

```
## S3 method for class 'lda.prdmt'
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.prmtd  
*predict.neuralnet.prmtd*

---

**Description**

Return prediction for a [neuralnet](#) model.

**Usage**

```
## S3 method for class 'neuralnet.prmtd'  
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.

---

predict.nnet.prmtd      *predict.nnet.prmtd*

---

### Description

Return prediction for a [nnet](#) model.

### Usage

```
## S3 method for class 'nnet.prmtd'
predict(object, newdata, type = "class", ...)
```

### Arguments

object	a <a href="#">nnet</a> 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 nnet model.

---

predict.qda.prmtd      *predict.qda.prmtd*

---

### Description

Return prediction for a [qda](#) model.

### Usage

```
## S3 method for class 'qda.prmtd'
predict(object, newdata, type = "class", ...)
```

### Arguments

object	a <a href="#">qda</a> 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 qda model.



---

```
predict.randomForest.prmtdt
  predict.randomForest.prmtdt
```

---

## Description

Return prediction for a [randomForest](#) model.

## Usage

```
## S3 method for class 'randomForest.prmtdt'
predict(
  object,
  newdata,
  type = "class",
  norm.votes = TRUE,
  predict.all = FALSE,
  proximity = FALSE,
  nodes = FALSE,
  cutoff,
  ...
)
```

## Arguments

<code>object</code>	a <a href="#">randomForest</a> model object for which prediction is desired.
<code>newdata</code>	an optional data frame in which to look for variables with which to predict.
<code>type</code>	type of prediction 'prob' or 'class' (default).
<code>norm.votes</code>	Should the vote counts be normalized (i.e., expressed as fractions)? Ignored if <code>object\$type</code> is regression.
<code>predict.all</code>	Should the predictions of all trees be kept?
<code>proximity</code>	Should proximity measures be computed? An error is issued if <code>object\$type</code> is regression.
<code>nodes</code>	Should the terminal node indicators (an n by ntree matrix) be return? If so, it is in the "nodes" attribute of the returned object.
<code>cutoff</code>	(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 <code>forest\$cutoff</code> component of <code>object</code> (i.e., the setting used when running <code>randomForest</code> ).
<code>...</code>	additional arguments affecting the predictions produced.

## Value

a vector or matrix of predictions for randomforest model.

---

predict.rpart.prmtd    *predict.rpart.prmtd*

---

### Description

Return prediction for a [rpart](#) model.

### Usage

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

### Arguments

object	a <a href="#">rpart</a> 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).
na.action	a function to determine what should be done with missing values in newdata. The default is to pass them down the tree using surrogates in the way selected when the model was built. Other possibilities are na.omit and na.fail.
...	additional arguments affecting the predictions produced.

### Value

a vector or matrix of predictions for rpart model.

---

predict.svm.prmtd    *predict.svm.prmtd*

---

### Description

Return prediction for a [svm](#) model.

### Usage

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

**Arguments**

object	a <code>svm</code> 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 <code>na.omit</code> , which leads to rejection of cases with missing values on any required variable. An alternative is <code>na.fail</code> , 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.

---

predict.xgb.Booster.prmtd  
*predict.xgb.Booster*

---

**Description**

Return prediction for a `xgb.train` model.

**Usage**

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

**Arguments**

<code>object</code>	a <code>xgb.train</code> model object for which prediction is desired.
<code>newdata</code>	an optional data frame in which to look for variables with which to predict.
<code>type</code>	type of prediction 'prob' or 'class' (default).
<code>missing</code>	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).
<code>outputmargin</code>	whether the prediction should be returned in the form of original untransformed sum of predictions from boosting iterations' results. E.g., setting <code>outputmargin=TRUE</code> for logistic regression would result in predictions for log-odds instead of probabilities.
<code>ntreelimit</code>	Deprecated, use <code>iterationrange</code> instead.
<code>predleaf</code>	whether predict leaf index.
<code>predcontrib</code>	whether to return feature contributions to individual predictions (see Details).
<code>approxcontrib</code>	whether to use a fast approximation for feature contributions (see Details).
<code>predinteraction</code>	whether to return contributions of feature interactions to individual predictions (see Details).
<code>reshape</code>	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 <code>predleaf</code> , <code>predcontrib</code> , or <code>predinteraction</code> flags is <code>TRUE</code> .
<code>...</code>	additional arguments affecting the predictions produced.

**Value**

a vector or matrix of predictions for xgb model.

---

`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")
```

---

print.indexes.prmtd    *Printing prmdt index object*

---

**Description**

Printing prmdt index object

**Usage**

```
## S3 method for class 'indexes.prmtd'
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**

```
## 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**

```
## 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	<i>ROC.area</i>
----------	-----------------

---

**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	<i>ROC.plot</i>
----------	-----------------

---

**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)
```

---

scaler

*scaler*

---

**Description**

Returns a scaled data.frame.

**Usage**

```
scaler(df)
```

**Arguments**

df                    A data.frame only with numeric variables.

**Value**

A data.frame.



---

train.ada	<i>train.ada</i>
-----------	------------------

---

### 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 <code>rpart.control</code> . For stumps, use <code>rpart.control(maxdepth=1,cp=1,minsplit=0,xval=0)</code> . <code>maxdepth</code> controls the depth of trees, and <code>cp</code> controls the complexity of trees. The priors should also be fixed through the <code>parms</code> 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= <code>na.rpart</code> .

### Value

A object `ada.prmtdt` 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
```

---

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), $\alpha = 1/2 \ln((1-\text{err})/\text{err})$ is used. If 'Freund' $\alpha = \ln((1-\text{err})/\text{err})$ 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 $\alpha = \ln((1-\text{err})/\text{err}) + \ln(\text{nclasses}-1)$ .
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.prmtdt` 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,minsplrit = 2,
                      maxdepth = 30, mfinal = 10)

model
predict <- predict(object = model,testing,type = "class")
predict
```

---

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 <code>class ~ x1 + x2 + ...</code> . 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.prmtdt 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, repl = FALSE)
ttesting <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.bayes <- train.bayes(Infant.Mortality~., ttraining)
prediction <- predict(model.bayes, ttesting)
prediction
```

---

train.gbm	<i>train.gbm</i>
-----------	------------------

---

## Description

Provides a wrapping function for the [gbm](#).

## Usage

```
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,  
  cv.folds = 0,  
  keep.data = TRUE,  
  verbose = F,  
  class.stratify.cv = NULL,  
  n.cores = NULL  
)
```

## 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 randomness into the model fit.
train.fraction	The first <code>train.fraction * nrow(data)</code> 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 <code>cv.folds &gt; 1</code> then gbm, in addition to the usual fit, will perform a cross-validation, calculate an estimate of generalization error returned in <code>cv.error</code> .
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 <code>gbm.more</code> 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 <code>gbm.more</code> , then it uses <code>verbose</code> 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 <code>n.cores</code> is not specified by the user, it is guessed using the <code>detectCores</code> function in the parallel package.

**Value**

A object `gbm.prmtdt` 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
```

---

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.prmtd 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)
ttesting <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.glm <- train.glm(Infant.Mortality~.,ttraining, family = "gaussian")
prediction <- predict(model.glm, ttesting)
prediction
```

---

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.prmtdt 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,]
ttraining <- swiss[-sampl,]
model.glmnet <- train.glmnet(Infant.Mortality~., ttraining, family = "gaussian")
prediction <- predict(model.glmnet, 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.pmdt 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**

```
# 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
```

---

train.lda	<i>train.lda</i>
-----------	------------------

---

**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.prmtd` 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, repl = FALSE)
ttesting <- iris[sampl,]
ttraining <- iris[-sampl,]
model.lda <- train.lda(Species~., ttraining)
model.lda
prediction <- predict(model.lda, ttesting)
prediction
```

---

train.neuralnet	<i>train.neuralnet</i>
-----------------	------------------------

---

## 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 meaningful.

**Value**

A object neuralnet.prmtdt 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	<i>train.nnet</i>
------------	-------------------

---

**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 <code>class ~ x1 + x2 + ...</code>
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 <code>na.omit</code> , 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.pmdt` 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**

```

# Classification
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

# 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

*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.prmtdt` 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)
model.qda
prediction <- predict(model.qda, 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.prmtdt 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

```
# 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(model.rf, ttesting)
prediction
```

---

train.rpart	<i>train.rpart</i>
-------------	--------------------

---

### 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 <a href="#">rpart.control</a> .
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 <a href="#">rpart.control</a> may also be specified in the call to rpart. They are checked against the list of valid arguments.

**Value**

A object `rpart.prmtd` 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

*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.pmdt 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**

```
# 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
```

---

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. <code>xgb.train</code> accepts only an <code>xgb.DMatrix</code> as the input. <code>xgboost</code> , in addition, also accepts <code>matrix</code> , <code>dgCMatrix</code> , or name of a local data file.
nrounds	max number of boosting iterations.
watchlist	named list of <code>xgb.DMatrix</code> datasets to use for evaluating model performance. Metrics specified in either <code>eval_metric</code> or <code>feval</code> will be computed for each of these datasets during each boosting iteration, and stored in the end as a field named <code>evaluation_log</code> in the resulting object. When either <code>verbose&gt;=1</code> or <code>cb.print.evaluation</code> callback is engaged, the performance results are continuously printed out during the training. E.g., specifying <code>watchlist=list(validation1=mat1, validation2=mat2)</code> allows to track the performance of each round's model on <code>mat1</code> and <code>mat2</code> .
obj	customized objective function. Returns gradient and second order gradient with given prediction and <code>dtrain</code> .
feval	customized evaluation function. Returns <code>list(metric='metric-name', value='metric-value')</code> with given prediction and <code>dtrain</code> .
verbose	If 0, <code>xgboost</code> will stay silent. If 1, it will print information about performance. If 2, some additional information will be printed out. Note that setting <code>verbose &gt; 0</code> automatically engages the <code>cb.print.evaluation(period=1)</code> callback function.
print_every_n	Print each n-th iteration evaluation messages when <code>verbose&gt;0</code> . Default is 1 which means all messages are printed. This parameter is passed to the <code>cb.print.evaluation</code> callback.
early_stopping_rounds	If <code>NULL</code> , the early stopping function is not triggered. If set to an integer <code>k</code> , training with a validation set will stop if the performance doesn't improve for <code>k</code> rounds. Setting this parameter engages the <code>cb.early.stop</code> 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 <a href="http://xgboost.readthedocs.io/en/latest">http://xgboost.readthedocs.io/en/latest</a>
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.prmtdt` 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 = 10, 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)
sampl <- sample(x = 1:len, size = len*0.20, replace = FALSE)
ttesting <- swiss[sampl,]
ttraining <- swiss[-sampl,]
model.xgb <- train.xgboost(Infant.Mortality~., ttraining, nrounds = 10, maximize = FALSE)
```

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

---

traineR

*Predictive (Classification and Regression) Models Homologator*

---

## Description

Methods to unify the different ways of creating predictive models and their different predictive formats for classification and regression. It includes methods such as K-Nearest Neighbors Schliep, K. P. (2004) <doi:10.5282/ubm/epub.1769>, Decision Trees Leo Breiman, Jerome H. Friedman, Richard A. Olshen, Charles J. Stone (2017) <doi:10.1201/9781315139470>, ADA Boosting Esteban Alfaro, Matias Gamez, Noelia García (2013) <doi:10.18637/jss.v054.i02>, Extreme Gradient Boosting Chen & Guestrin (2016) <doi:10.1145/2939672.2939785>, Random Forest Breiman (2001) <doi:10.1023/A:1010933404324>, Neural Networks Venables, W. N., & Ripley, B. D. (2002) <ISBN:0-387-95457-0>, Support Vector Machines Bennett, K. P. & Campbell, C. (2000) <doi:10.1145/380995.380999>, Bayesian Methods Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. (1995) <doi:10.1201/9780429258411>, Linear Discriminant Analysis Venables, W. N., & Ripley, B. D. (2002) <ISBN:0-387-95457-0>, Quadratic Discriminant Analysis Venables, W. N., & Ripley, B. D. (2002) <ISBN:0-387-95457-0>, Logistic Regression Dobson, A. J., & Barnett, A. G. (2018) <doi:10.1201/9781315182780> and Penalized Logistic Regression Friedman, J. H., Hastie, T., & Tibshirani, R. (2010) <doi:10.18637/jss.v033.i01>.

## Details

Package: traineR  
Type: Package  
Version: 2.2.2  
Date: 2025-05-23  
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

**See Also**

Useful links:

- <https://promidat.website/>
- Report bugs at <https://github.com/PROMiDAT/traineR/issues>

---

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

- \* **package**
  - traineR, 51
- ada, 10, 25
- boosting, 8, 10, 11, 26, 27
- categorical.predictive.power, 3
- confusion.matrix, 4
- contr.dummy, 5
- contr.metric, 5
- contr.ordinal, 6
- cv.glmnet, 14
  
- gbm, 12, 29, 30
- general.indexes, 6
- ggplot, 4, 8, 9, 21
- glm, 12, 13, 31, 33
- glmnet, 13, 14, 33, 34
  
- importance.plot, 7
  
- lda, 14, 15, 37
  
- naiveBayes, 11, 27, 28
- neuralnet, 15, 38–40
- nnet, 16, 40
- numerical.predictive.power, 8
  
- performance, 23, 24
- plot.prmtd, 9
- predict.ada.prmtd, 10
- predict.adabag.prmtd, 10
- predict.bayes.prmtd, 11
- predict.gbm.prmtd, 12
- predict.glm.prmtd, 12
- predict.glmnet.prmtd, 13
- predict.knn.prmtd, 14
- predict.lda.prmtd, 14
- predict.neuralnet.prmtd, 15
- predict.nnet.prmtd, 16
  
- predict.qda.prmtd, 16
- predict.randomForest.prmtd, 17
- predict.rpart.prmtd, 18
- predict.svm.prmtd, 18
- predict.xgb.Booster.prmtd, 19
- prediction, 23, 24
- prediction.variable.balance, 20
- print.indexes.prmtd, 21
- print.prediction.prmtd, 22
- print.prmtd, 22
  
- qda, 16, 41, 42
  
- randomForest, 17, 42, 43
- ROC.area, 23
- ROC.plot, 23
- rpart, 18, 44, 45
- rpart.control, 27, 45
  
- scaler, 24
- svm, 18, 19, 46, 47
  
- train.ada, 25
- train.adabag, 8, 26
- train.bayes, 27
- train.gbm, 29
- train.glm, 31
- train.glmnet, 33
- train.kknn, 5, 6, 14, 35, 36
- train.knn, 35
- train.lda, 37
- train.neuralnet, 38
- train.nnet, 40
- train.qda, 41
- train.randomForest, 42
- train.rpart, 44
- train.svm, 46
- train.xgboost, 47
- traineR, 51
- traineR-package (traineR), 51

`varplot`, [52](#)

`xgb.train`, [19](#), [20](#), [47](#), [50](#)