Package 'mlearning'

April 26, 2022

Type Package

Title Machine Learning Algorithms with Unified Interface and Confusion Matrices

Version 1.1.1

Date 2022-04-26

Author Philippe Grosjean [aut, cre], Kevin Denis [aut]

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

Depends R (>= 3.0.4)

Imports stats, grDevices, class, nnet, MASS, e1071, randomForest, ipred

Suggests mlbench, datasets, RColorBrewer

Description A unified interface is provided to various machine learning algorithms like LDA, QDA, k-nearest neighbour, LVQ, random forest, SVM, ... It allows to train, test, and apply cross-validation using similar functions and function arguments with a minimalist and clean, formula-based interface. Missing data are threated the same way as base and stats R functions for all algorithms, both in training and testing. Confusion matrices are also provided with a rich set of metrics calculated and a few specific plots.

License GPL (>= 2)

URL https://www.sciviews.org/mlearning/

NeedsCompilation no

Repository CRAN

Date/Publication 2022-04-26 11:40:05 UTC

R topics documented:

mlearning-package 2
confusion 2
mlearning 6

Index **15**

`mlearning-package` *mlearning: machine learning algorithms with a common UI and confusion matrices*

Description

This package provides wrappers around several existing machine learning algorithms in R, under a unified user interface. Confusion matrices can also be calculated and viewed as tables or plots.

Details

Package:	mlearning
Type:	Package
Version:	1.0.3
Date:	2017-12-12
License:	GPL 2 or above at your convenience.

Author(s)

Philippe Grosjean & Kevin Denis, Numerical Ecology of Aquatic Systems, Mons University, Belgium.

Maintainer: Philippe Grosjean <Philippe.Grosjean@umons.ac.be>

See Also

`mlearning`, `confusion`

`confusion` *Construct and analyze confusion matrices*

Description

Confusion matrices compare two classifications (usually one done automatically using a machine learning algorithm versus the true classication represented by a manual classification by a specialist... but one can also compare two automatic or two manual classifications against each other).

Usage

```
confusion(x, ...)
## Default S3 method:
confusion(x, y = NULL, vars = c("Actual", "Predicted"),
    labels = vars, merge.by = "Id", useNA = "ifany", prior, ...)
## S3 method for class 'mlearning'
confusion(x, y = response(x),
    labels = c("Actual", "Predicted"), useNA = "ifany", prior, ...)

## S3 method for class 'confusion'
print(x, sums = TRUE, error.col = sums, digits = 0,
    sort = "ward.D2", ...)
## S3 method for class 'confusion'
summary(object, type = "all", sort.by = "Fscore",
    decreasing = TRUE, ...)
## S3 method for class 'summary.confusion'
print(x, ...)
## S3 method for class 'confusion'
plot(x, y = NULL, type = c("image", "barplot", "stars",
    "dendrogram"), stat1 = "Recall", stat2 = "Precision", names, ...)

confusionImage(x, y = NULL, labels = names(dimnames(x)), sort = "ward.D2",
    numbers = TRUE, digits = 0, mar = c(3.1, 10.1, 3.1, 3.1), cex = 1, asp = 1,
    colfun, ncols = 41, col0 = FALSE, grid.col = "gray", ...)
confusionBarplot(x, y = NULL, col = c("PeachPuff2", "green3", "lemonChiffon2"),
    mar = c(1.1, 8.1, 4.1, 2.1), cex = 1, cex.axis = cex, cex.legend = cex,
    main = "F-score (precision versus recall)", numbers = TRUE, min.width = 17,
    ...)
confusionStars(x, y = NULL, stat1 = "Recall", stat2 = "Precision", names, main,
    col = c("green2", "blue2", "green4", "blue4"), ...)
confusionDendrogram(x, y = NULL, labels = rownames(x), sort = "ward.D2",
    main = "Groups clustering", ...)

prior(object, ...)
## S3 method for class 'confusion'
prior(object, ...)
prior(object, ...) <- value
## S3 replacement method for class 'confusion'
prior(object, ...) <- value
```

Arguments

`x`	an object.
`y`	another object, from which to extract the second classification, or `NULL` if not used.
`vars`	the variables of interest in the first and second classification in the case the objects are lists or data frames. Otherwise, this argument is ignored and `x` and `y` must be factors with same length and same levels.

`labels`	labels to use for the two classifications. By default, it is the same as `vars` or the one in the confusion matrix.
`merge.by`	a character string with the name of variables to use to merge the two data frames, or `NULL`.
`useNA`	do we keep NAs as a separate category? The default `"ifany"` creates this category only if there are missing values. Other possibilities are `"no"`, or `"always"`.
`prior`	class frequencies to use for first classifier that is tabulated in the rows of the confusion matrix. For its value, see here under, the `value` argument.
`sums`	is the confusion matrix printed with rows and columns sums?
`error.col`	is a column with class error for first classifier added (equivalent to flase negative rate of FNR)?
`digits`	the number of digits after the decimal point to print in the confusion matrix. The default or zero leads to most compact presentation and is suitable for frequencies, but not for relative frequencies.
`sort`	are rows and columns of the confusion matrix sorted so that classes with larger confusion are closer together? Sorting is done using a hierachical clustering with `hclust()`. The clustering method is provided is the one provides (`"ward.D2"`, by default, but see the `hclust()` help for other options). If `FALSE` or `NULL`, no sorting is done.
`object`	a 'confusion' object.
`sort.by`	the statistics to use to sort the table (by default, Fmeasure, the F1 score for each class = 2 * recall * precision / (recall + precision)).
`decreasing`	do we sort in increasing or decreasing order?
`type`	the type of graph to plot (only `"stars"` if two confusion matrices are to be compared).
`stat1`	first statistic to compare in the stars plot.
`stat2`	second statistic to compare in the stars plot.
`...`	further arguments passed to the function. In particular for `plot()`, it can be all arguments for the corresponding plot.
`numbers`	are actual numbers indicated in the confusion matrix image?
`mar`	graph margins.
`cex`	text magnification factor.
`cex.axis`	idem for axes. If `NULL`, the axis is not drawn.
`cex.legend`	idem for legend text. If `NULL`, no legend is added.
`asp`	graph aspect ration. There is little reasons to cvhange the default value of 1.
`col`	color(s) to use fir the graph.
`colfun`	a function that calculates a series of colors, like e.g., `cm.colors()` and that accepts one argument being the number of colors to be generated.
`ncols`	the number of colors to generate. It should preferrably be 2 * number of levels + 1, where levels is the number of frequencies you want to evidence in the plot. Default to 41.

`col0`	should null values be colored or not (no, by default)?
`grid.col`	color to use for grid lines, or `NULL` for not drawing grid lines.
`names`	names of the two classifiers to compare.
`main`	main title of the graph.
`min.width`	minimum bar width required to add numbers.
`value`	a single positive numeric to set all class frequencies to this value (use 1 for relative frequencies and 100 for relative freqs in percent), or a vector of positive numbers of the same length as the levels in the object. If the vector is named, names must match levels. Alternatively, providing `NULL` or an object of null length resets row class frequencies into their initial values.

Value

A confusion matrix in a 'confusion' object. `prior()` returns the current class frequencies associated with first classification tabulated, i.e., for rows in the confusion matrix.

Author(s)

Philippe Grosjean <Philippe.Grosjean@umons.ac.be> and Kevin Denis <Kevin.Denis@umons.ac.be>

See Also

`mlearning`, `hclust`, `cm.colors`

Examples

```
data("Glass", package = "mlbench")
## Use a little bit more informative labels for Type
Glass$Type <- as.factor(paste("Glass", Glass$Type))

## Use learning vector quantization to classify the glass types
## (using default parameters)
summary(glassLvq <- mlLvq(Type ~ ., data = Glass))

## Calculate cross-validated confusion matrix and plot it in different ways
(glassConf <- confusion(cvpredict(glassLvq), Glass$Type))
## Raw confusion matrix: no sort and no margins
print(glassConf, sums = FALSE, sort = FALSE)
## Graphs
plot(glassConf) # Image by default
plot(glassConf, sort = FALSE) # No sorting
plot(glassConf, type = "barplot")
plot(glassConf, type = "stars")
plot(glassConf, type = "dendrogram")

summary(glassConf)
summary(glassConf, type = "Fscore")

## Build another classifier and make a comparison
summary(glassNaiveBayes <- mlNaiveBayes(Type ~ ., data = Glass))
```

```
(glassConf2 <- confusion(cvpredict(glassNaiveBayes), Glass$Type))

## Comparison plot for two classifiers
plot(glassConf, glassConf2)

## When the probabilities in each class do not match the proportions in the
## training set, all these calculations are useless. Having an idea of
## the real proportions (so-called, priors), one should first reweight the
## confusion matrix before calculating statistics, for instance:
prior1 <- c(10, 10, 10, 100, 100, 100) # Glass types 1-3 are rare
prior(glassConf) <- prior1
glassConf
summary(glassConf, type = c("Fscore", "Recall", "Precision"))
plot(glassConf)

## This is very different than if glass types 1-3 are abundants!
prior2 <- c(100, 100, 100, 10, 10, 10) # Glass types 1-3 are abundants
prior(glassConf) <- prior2
glassConf
summary(glassConf, type = c("Fscore", "Recall", "Precision"))
plot(glassConf)

## Weight can also be used to construct a matrix of relative frequencies
## In this case, all rows sum to one
prior(glassConf) <- 1
print(glassConf, digits = 2)
## However, it is easier to work with relative frequencies in percent
## and one gets a more compact presentation
prior(glassConf) <- 100
glassConf

## To reset row class frequencies to original propotions, just assign NULL
prior(glassConf) <- NULL
glassConf
prior(glassConf)
```

`mlearning` *Alternate interface to various machine learning algorithms*

Description

In order to provide a unified (formula-based) interface to various machine learning algorithms, these function wrap a common UI around a couple of existing code.

Usage

```
mlearning(formula, data, method, model.args, call = match.call(), ...,
    subset, na.action = na.fail)
## S3 method for class 'mlearning'
print(x, ...)
```

```
## S3 method for class 'mlearning'
summary(object, ...)
## S3 method for class 'summary.mlearning'
print(x, ...)
## S3 method for class 'mlearning'
plot(x, y, ...)
## S3 method for class 'mlearning'
predict(object, newdata, type = c("class", "membership", "both"),
    method = c("direct", "cv"), na.action = na.exclude, ...)

cvpredict(object, ...)
## S3 method for class 'mlearning'
cvpredict(object, type = c("class", "membership", "both"),
    cv.k = 10, cv.strat = TRUE, ...)

mlLda(train, ...)
## Default S3 method:
mlLda(train, response, ...)
## S3 method for class 'formula'
mlLda(formula, data, ..., subset, na.action)
## S3 method for class 'mlLda'
predict(object, newdata, type = c("class", "membership", "both",
    "projection"), prior = object$prior, dimension,
    method = c("plug-in", "predictive", "debiased", "cv"), ...)

mlQda(train, ...)
## Default S3 method:
mlQda(train, response, ...)
## S3 method for class 'formula'
mlQda(formula, data, ..., subset, na.action)
## S3 method for class 'mlQda'
predict(object, newdata, type = c("class", "membership", "both"),
    prior = object$prior, method = c("plug-in", "predictive", "debiased",
    "looCV", "cv"), ...)

mlRforest(train, ...)
## Default S3 method:
mlRforest(train, response, ntree = 500, mtry, replace = TRUE, classwt = NULL, ...)
## S3 method for class 'formula'
mlRforest(formula, data, ntree = 500, mtry, replace = TRUE, classwt = NULL, ...,
    subset, na.action)
## S3 method for class 'mlRforest'
predict(object, newdata, type = c("class", "membership", "both",
    "vote"), method = c("direct", "oob", "cv"), ...)

mlNnet(train, ...)
## Default S3 method:
mlNnet(train, response, size = NULL, rang = NULL, decay = 0, maxit = 1000, ...)
```

```
## S3 method for class 'formula'
mlNnet(formula, data, size = NULL, rang = NULL, decay = 0, maxit = 1000, ...,
    subset, na.action)
## S3 method for class 'mlNnet'
predict(object, newdata, type = c("class", "membership", "both", "raw"),
    method = c("direct", "cv"), na.action = na.exclude,...)

mlKnn(train, ...)
## Default S3 method:
mlKnn(train, response, k.nn = 5, ...)
## S3 method for class 'formula'
mlKnn(formula, data, k.nn = 5, ..., subset, na.action)
## S3 method for class 'mlKnn'
predict(object, newdata, type = c("class", "prob", "both"),
    method = c("direct", "cv"), na.action = na.exclude,...)

mlLvq(train, ...)
## Default S3 method:
mlLvq(train, response, k.nn = 5, size, prior, algorithm = "olvq1", ...)
## S3 method for class 'formula'
mlLvq(formula, data, k.nn = 5, size, prior, algorithm = "olvq1", ...,
    subset, na.action)
## S3 method for class 'lvq'
summary(object, ...)
## S3 method for class 'summary.lvq'
print(x, ...)
## S3 method for class 'mlLvq'
predict(object, newdata, type = "class", method = c("direct", "cv"),
    na.action = na.exclude,...)

mlSvm(train, ...)
## Default S3 method:
mlSvm(train, response, scale = TRUE, type = NULL, kernel = "radial",
    classwt = NULL, ...)
## S3 method for class 'formula'
mlSvm(formula, data, scale = TRUE, type = NULL, kernel = "radial",
    classwt = NULL, ..., subset, na.action)
## S3 method for class 'mlSvm'
predict(object, newdata, type = c("class", "membership", "both"),
    method = c("direct", "cv"), na.action = na.exclude, ...)

mlNaiveBayes(train, ...)
## Default S3 method:
mlNaiveBayes(train, response, laplace = 0, ...)
## S3 method for class 'formula'
mlNaiveBayes(formula, data, laplace = 0, ..., subset, na.action)
## S3 method for class 'mlNaiveBayes'
predict(object, newdata, type = c("class", "membership", "both"),
    method = c("direct", "cv"), na.action = na.exclude, threshold = 0.001, eps = 0, ...)

response(object, ...)
## Default S3 method:
response(object, ...)
train(object, ...)
## Default S3 method:
train(object, ...)
```

Arguments

`formula`	a formula with left term being the factor variable to predict (for supervised classification), a vector of numbers (for regression) or nothing (for unsupervised classification) and the right term with the list of independent, predictive variables, separated with a plus sign. If the data frame provided contains only the dependent and independent variables, one can use the `class ~ .` short version (and that one is strongly encouraged). Variables with minus sign are eliminated and calculations on variables are possible according to usual formula convention (possibly protected by `I()`).
`data`	a data.frame to use as a training set.
`method`	a machine learning method to use. For `predict()`, it is the prediction method. According to `predict.lda` in package MASS: determines how the parameter estimation is handled. With `"plug-in"` (the default) the usual unbiased parameter estimates are used and assumed to be correct. With `"debiased"` an unbiased estimator of the log posterior probabilities is used, and with `"predictive"` the parameter estimates are integrated out using a vague prior. With `"looCV"` the leave-one-out cross-validation fits to the original dataset are computed and returned. If you indicate `method = "cv"` then `cvpredict()` is used and you cannot provide `newdata` in that case.
`model.args`	arguments for formula modeling with substituted data and subset... Not to be used by the end-user.
`call`	the function call. Not to be used by the end-user.
`...`	further arguments passed to the machine learning algorithm or the `predict()` method. See original algorithm.
`subset`	index vector with the cases to define the training set in use (this argument must be named, if provided).
`na.action`	function to specify the action to be taken if NAs are found `na.fail`, by default. Another option is `na.omit`, and cases with missing values on any required variable are dropped (this argument must be named, if provided). The default, and most suitable option for `predict()` methods is `na.exclude` and rows with NAs in newdata are then, excluded from prediction, but reinjected in final results.
`cv.k`	k for k-fold cross validation, cf `errorest()`.
`cv.strat`	is the subsampling stratified or not in cross validation, cf `errorest()`.
`x`	a mlearning object.
`y`	another object (depending on the machine learning algorithm, but it is usually not used).

`object`	one of the mlearning objects.
`newdata`	a data.frame with same variables as `data` to use for new predictions.
`type`	the type of result to get. Usually, `"class"`, which is the default. Depending on the algorithm, other types are also available. `membership` and `both` are almost always available too. `membership` corresponds to posterior probability, raw results, normalized votes, etc., depending on the machine learning algorithm. With `both`, class and membership are both returned at once in a list. For `mlSvm()`, it is the type of algorithm to use (see `?svm`).
`train`	a matrix or data frame with predictors.
`response`	a vector of factor (classification) or numeric (regression), or `NULL` (unsupervised classification).
`prior`	prior probabilities of the classes (the proportions in the training set are used by default). For `mlLvq()`, probabilities to represent classes in the codebook (default values are the proportions in the training set).
`dimension`	the dimension of the space to be used for prediction.
`ntree`	the number of trees to generate (use a value large enough to get at least a few predictions for each input row).
`mtry`	number of variables randomly sampled as candidates at each split.
`replace`	sample cases with or without replacement?
`classwt`	priors of the classes. Need not add up to one.
`size`	number of units in the hidden layer for `mlNnet()`. Can be zero if there are skip-layer units. If `NULL`, a reasonalbe default is computed. for `mlLvq()`, the size of the codebook. Defaults to min(round(0.4*ng*(ng-1 + p/2),0), n) where ng is the number of classes.
`rang`	initial random weights on [-rang, rang]. Value about 0.5 unless the inputs are large, in which case it should be chosen so that rang * max(|x|) is about 1. If `NULL`, a reasonalbe default is computed.
`decay`	parameter for weight decay. Default 0.
`maxit`	maximum number of iterations. Default 1000.
`k.nn`	k used for k-NN number of neighbour considered. Default is 5.
`algorithm`	an algorithm among 'olvq1' (default, the optimized lvq1), 'lvq1', 'lvq2', or 'lvq3'.
`scale`	are all the variables scaled? If a vector is provided, it is applied to variables with recycling.
`kernel`	the kernel used by svm, see `?svm`. Can be "radial", "linear", "polynomial" or "sigmoid".
`laplace`	positive double controlling Laplace smoothing for the naive Bayes classifier. The default (0) disables Laplace smoothing.
`threshold`	Value replacing cells with probabilities within 'eps' range for Naive Bayes predictions.
`eps`	double for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by 'theshold') for Naive Bayes predictions.

Details

TODO: explain here the mechanism used to provide a common interface on top of various existing algorithms, and how one can add new items.

Value

A machine learning object where the `predict()` method can be applied to classify new items.

For `response()` and `train()`, the respective resmonse vector and training matrix (the matrix with all predicting terms).

Author(s)

All these functions are just wrapper around existing R code written by Philippe Grosjean <Philippe.Grosjean@umons.ac.be> in order to get similar interface and objects. All credits to original authors (click here under).

See Also

`confusion`, `errorest`, `lda`, `qda`, `randomForest`, `olvq1`, `nnet`, `naiveBayes`, `svm`

Examples

```
## Prepare data: split into training set (2/3) and test set (1/3)
data("iris", package = "datasets")
train <- c(1:34, 51:83, 101:133)
irisTrain <- iris[train, ]
irisTest <- iris[-train, ]
## One case with missing data in train set, and another case in test set
irisTrain[1, 1] <- NA
irisTest[25, 2] <- NA

data("HouseVotes84", package = "mlbench")

data(airquality, package = "datasets")

## Supervised classification using linear discriminant analysis
irLda <- mlLda(Species ~ ., data = irisTrain)
irLda
summary(irLda)
plot(irLda, col = as.numeric(response(irLda)) + 1)
predict(irLda, newdata = irisTest) # class (default type)
predict(irLda, type = "membership") # posterior probability
predict(irLda, type = "both") # both class and membership in a list
## Sometimes, other types are allowed, like for lda:
predict(irLda, type = "projection") # Projection on the LD axes
## Add test set items to the previous plot
points(predict(irLda, newdata = irisTest, type = "projection"),
    col = as.numeric(predict(irLda, newdata = irisTest)) + 1, pch = 19)

## predict() and confusion() should be used on a separate test set
## for unbiased estimation (or using cross-validation, bootstrap, ...)
confusion(irLda) # Wrong, cf. biased estimation (so-called, self-consistency)
```

```
## Estimation using a separate test set
confusion(predict(irLda, newdata = irisTest), irisTest$Species)

## Another dataset (binary predictor... not optimal for lda, just for test)
summary(res <- mlLda(Class ~ ., data = HouseVotes84, na.action = na.omit))
confusion(res) # Self-consistency
print(confusion(res), error.col = FALSE) # Without error column

## More complex formulas
summary(mlLda(Species ~ . - Sepal.Width, data = iris)) # Exclude variable
summary(mlLda(Species ~ log(Petal.Length) + log(Petal.Width) +
    I(Petal.Length/Sepal.Length), data = iris)) # With calculations

## Factor levels with missing items are allowed
ir2 <- iris[-(51:100), ] # No Iris versicolor in the training set
summary(res <- mlLda(Species ~ ., data = ir2)) # virginica is NOT there
## Missing levels are reinjected in class or membership by predict()
predict(res, type = "both")
## ... but, of course, the classifier is wrong for Iris versicolor
confusion(predict(res, newdata = iris), iris$Species)

## Simpler interface, but more memory-effective
summary(mlLda(train = iris[, -5], response = iris$Species))


## Supervised classification using quadratic discriminant analysis
summary(res <- mlQda(Species ~ ., data = irisTrain))
confusion(res) # Self-consistency
confusion(predict(res, newdata = irisTest), irisTest$Species) # Performances

## Another dataset (binary predictor... not optimal for qda, just for test)
summary(res <- mlQda(Class ~ ., data = HouseVotes84, na.action = na.omit))
confusion(res) # Self-consistency


## Supervised classification using random forest
summary(res <- mlRforest(Species ~ ., data = irisTrain))
plot(res)
## For such a relatively simple case, 50 trees are enough
summary(res <- mlRforest(Species ~ ., data = irisTrain, ntree = 50))
predict(res) # Default type is class
predict(res, type = "membership")
predict(res, type = "both")
predict(res, type = "vote")
## Out-of-bag prediction
predict(res, method = "oob")
confusion(res) # Self-consistency
confusion(res, method = "oob") # Out-of-bag performances
## Cross-validation prediction is a good choice when there is no test set:
predict(res, method = "cv")  # Idem: cvpredict(res)
confusion(res, method = "cv") # Cross-validation for performances estimation
## Evaluation of performances using a separate test set
confusion(predict(res, newdata = irisTest), irisTest$Species) # Test set perfs
```

```
## Regression using random forest (from ?randomForest)
set.seed(131)
summary(ozone.rf <- mlRforest(Ozone ~ ., data = airquality, mtry = 3,
    importance = TRUE, na.action = na.omit))
## Show "importance" of variables: higher value mean more important:
round(randomForest::importance(ozone.rf), 2)
plot(na.omit(airquality)$Ozone, predict(ozone.rf))
abline(a = 0, b = 1)

## Unsupervised classification using random forest (from ?randomForest)
set.seed(17)
summary(iris.urf <- mlRforest(~ ., iris[, -5]))
randomForest::MDSplot(iris.urf, iris$Species)
plot(hclust(as.dist(1 - iris.urf$proximity), method = "average"),
    labels = iris$Species)


## Supervised classification using neural network
set.seed(689)
summary(res <- mlNnet(Species ~ ., data = irisTrain))
predict(res) # Default type is class
predict(res, type = "membership")
predict(res, type = "both")
confusion(res) # Self-consistency
confusion(predict(res, newdata = irisTest), irisTest$Species) # Test set perfs

## Idem, but two classes prediction using factor predictors
set.seed(325)
summary(res <- mlNnet(Class ~ ., data = HouseVotes84, na.action = na.omit))
confusion(res) # Self-consistency

## Regression using neural network
set.seed(34)
summary(ozone.nnet <- mlNnet(Ozone ~ ., data = airquality, na.action = na.omit,
    skip = TRUE, decay = 1e-3, size = 20, linout = TRUE))
plot(na.omit(airquality)$Ozone, predict(ozone.nnet, type = "raw"))
abline(a = 0, b = 1)


## Supervised classification using k-nearest neighbours
summary(res <- mlKnn(Species ~ ., data = irisTrain))
predict(res) # This object only returns class
confusion(res) # Self-consistency
confusion(predict(res, newdata = irisTest), irisTest$Species) # Test set perfs


## Supervised classification using learning vector quantization
summary(res <- mlLvq(Species ~ ., data = irisTrain))
predict(res) # This object only returns class
confusion(res) # Self-consistency
confusion(predict(res, newdata = irisTest), irisTest$Species) # Test set perfs
```

```
## Supervised classification using support vector machine
summary(res <- mlSvm(Species ~ ., data = irisTrain))
predict(res) # Default type is class
predict(res, type = "membership")
predict(res, type = "both")
confusion(res) # Self-consistency
confusion(predict(res, newdata = irisTest), irisTest$Species) # Test set perfs

## Another dataset
summary(res <- mlSvm(Class ~ ., data = HouseVotes84, na.action = na.omit))
confusion(res) # Self-consistency

## Regression using support vector machine
summary(ozone.svm <- mlSvm(Ozone ~ ., data = airquality, na.action = na.omit))
plot(na.omit(airquality)$Ozone, predict(ozone.svm))
abline(a = 0, b = 1)


## Supervised classification using naive Bayes
summary(res <- mlNaiveBayes(Species ~ ., data = irisTrain))
predict(res) # Default type is class
predict(res, type = "membership")
predict(res, type = "both")
confusion(res) # Self-consistency
confusion(predict(res, newdata = irisTest), irisTest$Species) # Test set perfs

## Another dataset
summary(res <- mlNaiveBayes(Class ~ ., data = HouseVotes84, na.action = na.omit))
confusion(res) # Self-consistency
```

Index

∗ **package**
    mlearning-package, 2
∗ **tree**
    confusion, 2
    mlearning, 6

cm.colors, *5*
confusion, *2*, 2, *11*
confusionBarplot *(*confusion*)*, 2
confusionDendrogram *(*confusion*)*, 2
confusionImage *(*confusion*)*, 2
confusionStars *(*confusion*)*, 2
cvpredict *(*mlearning*)*, 6

errorest, *11*

hclust, *5*

lda, *11*

mlearning, *2*, *5*, 6
mlearning-package, 2
mlKnn *(*mlearning*)*, 6
mlLda *(*mlearning*)*, 6
mlLvq *(*mlearning*)*, 6
mlNaiveBayes *(*mlearning*)*, 6
mlNnet *(*mlearning*)*, 6
mlQda *(*mlearning*)*, 6
mlRforest *(*mlearning*)*, 6
mlSvm *(*mlearning*)*, 6

naiveBayes, *11*
nnet, *11*

olvq1, *11*

plot.confusion *(*confusion*)*, 2
plot.mlearning *(*mlearning*)*, 6
predict.mlearning *(*mlearning*)*, 6
predict.mlKnn *(*mlearning*)*, 6
predict.mlLda *(*mlearning*)*, 6
predict.mlLvq *(*mlearning*)*, 6
predict.mlNaiveBayes *(*mlearning*)*, 6
predict.mlNnet *(*mlearning*)*, 6
predict.mlQda *(*mlearning*)*, 6
predict.mlRforest *(*mlearning*)*, 6
predict.mlSvm *(*mlearning*)*, 6
print.confusion *(*confusion*)*, 2
print.mlearning *(*mlearning*)*, 6
print.summary.confusion *(*confusion*)*, 2
print.summary.lvq *(*mlearning*)*, 6
print.summary.mlearning *(*mlearning*)*, 6
prior *(*confusion*)*, 2
prior<- *(*confusion*)*, 2

qda, *11*

randomForest, *11*
response *(*mlearning*)*, 6

summary.confusion *(*confusion*)*, 2
summary.lvq *(*mlearning*)*, 6
summary.mlearning *(*mlearning*)*, 6
svm, *11*

train *(*mlearning*)*, 6