Package 'SLmetrics'

March 18, 2025

Title Machine Learning Performance Evaluation on Steroids

Version 0.3-3

Description Performance evaluation metrics for supervised and unsupervised machine learning, statistical learning and artificial intelligence applications. Core computations are implemented in 'C++' for scalability and efficiency.

```
License GPL (>= 3)
Encoding UTF-8
RoxygenNote 7.3.2
LinkingTo Rcpp, RcppEigen
Suggests knitr, reticulate, rmarkdown, testthat (>= 3.0.0)
Config/testthat/edition 3
Imports grDevices, lattice, Rcpp
Depends R (>= 4.3)
URL https://serkor1.github.io/SLmetrics/,
     https://github.com/serkor1/SLmetrics
BugReports https://github.com/serkor1/SLmetrics/issues
LazyData true
VignetteBuilder knitr
NeedsCompilation yes
Author Serkan Korkmaz [cre, aut, cph]
     (<https://orcid.org/0000-0002-5052-0982>)
Maintainer Serkan Korkmaz < serkor 1@duck.com>
Repository CRAN
Date/Publication 2025-03-18 15:20:10 UTC
```

2 Contents

Contents

accuracy.factor
auc.numeric
baccuracy.factor
banknote
ccc.numeric
ckappa.factor
cmatrix.factor
dor.factor
entropy.matrix
fbeta.factor
fdr.factor
fer.factor
fmi.factor
fpr.factor
huberloss.numeric
jaccard.factor
logloss.factor
mae.numeric
mape.numeric
mcc.factor
mpe.numeric
mse.numeric
nlr.factor
npv.factor
obesity
openmp.on
pinball.numeric
plr.factor
pr.auc.matrix
precision.factor
preorder
presort
prROC.factor
rae.numeric
recall.factor
rmse.numeric
rmsle.numeric
roc.auc.matrix
ROC.factor
rrmse.numeric
rrse.numeric
rsq.numeric
smape.numeric
specificity.factor
wine_quality
zerooneloss factor

accuracy.factor 3

Index 133

```
accuracy.factor Accuracy
```

Description

A generic function for the (normalized) accuracy in classification tasks. Use weighted.accuracy() for the weighted accuracy.

Usage

```
## S3 method for class 'factor'
accuracy(actual, predicted, ...)

## S3 method for class 'factor'
weighted.accuracy(actual, predicted, w, ...)

## S3 method for class 'cmatrix'
accuracy(x, ...)

## Generic S3 method
accuracy(...)

## Generic S3 method
weighted.accuracy(
...,
w
)
```

Arguments

actual	A vector of \prec factor \gt with length n , and k levels
predicted	A vector of <factor> with length n, and k levels</factor>
	Arguments passed into other methods
W	A <numeric>-vector of length n. NULL by default</numeric>
х	A confusion matrix created cmatrix()

Value

A <numeric>-vector of length 1

4 accuracy.factor

Definition

Let $\hat{\alpha} \in [0,1]$ be the proportion of correctly predicted classes. The accuracy of the classifier is calculated as,

$$\hat{\alpha} = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN}$$

Where:

- #TP is the number of true positives,
- #TN is the number of true negatives,
- #FP is the number of false positives, and
- #FN is the number of false negatives.

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

accuracy.factor 5

See Also

```
Other Classification: ROC.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rsg.numeric(), rmsle.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
        = iris,
  family = binomial(
    link = "logit"
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
    predict(model, type = "response") > 0.5
  ),
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
 levels = c(1,0),
  labels = c("Virginica", "Others")
)
```

6 auc.numeric

```
# 4) evaluate model
# performance
cat(
  "Accuracy", accuracy(
   actual
            = actual,
   predicted = predicted
 ),
  "Accuracy (weigthed)", weighted.accuracy(
            = actual,
   actual
   predicted = predicted,
             = iris$Petal.Length/mean(iris$Petal.Length)
 ),
 sep = "\n"
)
```

auc.numeric

AUC

Description

The auc()-function calculates the area under the curve.

Usage

```
## S3 method for class 'numeric'
auc(y, x, method = 0L, presorted = TRUE, ...)
## Generic S3 method
auc(
   y,
   x,
   method = 0,
   presorted = TRUE,
   ...
)
```

Arguments

y A <numeric> vector of length n. x A <numeric> vector of length n.

method A <numeric> value (default: 0). Defines the underlying method of calculating the area under the curve. If 0 it is calculated using the trapezoid-method, if 1

it is calculated using the step-method.

presorted A <logical>-value length 1 (default: FALSE). If TRUE the input will not be

sorted by threshold.

... Arguments passed into other methods.

auc.numeric 7

Value

A < numeric > vector of length 1

Definition

Trapezoidal rule

The **trapezoidal rule** approximates the integral of a function f(x) between x = a and x = b using trapezoids formed between consecutive points. If we have points x_0, x_1, \ldots, x_n (with $a = x_0 < x_1 < \cdots < x_n = b$) and corresponding function values $f(x_0), f(x_1), \ldots, f(x_n)$, the area under the curve A_T is approximated by:

$$A_T \approx \sum_{k=1}^{n} \frac{f(x_{k-1}) + f(x_k)}{2} [x_k - x_{k-1}].$$

Step-function method

The **step-function** (**rectangular**) **method** uses the value of the function at one endpoint of each subinterval to form rectangles. With the same partition x_0, x_1, \ldots, x_n , the rectangular approximation A_S can be written as:

$$A_S \approx \sum_{k=1}^{n} f(x_{k-1}) [x_k - x_{k-1}].$$

See Also

Other Tools: cov.wt.matrix(), preorder(), presort()

```
## 1) Ordered x and y pair
x \leftarrow seq(0, pi, length.out = 200)
y < -\sin(x)
## 1.1) calculate area
ordered_auc <- auc(y = y, x = x)
## 2) Unordered x and y pair
x \leftarrow sample(seq(0, pi, length.out = 200))
y < -\sin(x)
## 2.1) calculate area
unordered_auc <- auc(y = y, x = x)
## 2.2) calculate area with explicit
## ordering
unordered_auc_flag <- auc(</pre>
  y = y,
 x = x,
  presorted = FALSE
)
```

8 baccuracy.factor

```
## 3) display result
cat(
   "AUC (ordered x and y pair)", ordered_auc,
   "AUC (unordered x and y pair)", unordered_auc,
   "AUC (unordered x and y pair, with unordered flag)", unordered_auc_flag,
   sep = "\n"
)
```

baccuracy.factor

Balanced Accuracy

Description

A generic function for the (normalized) balanced accuracy. Use weighted.baccuracy() for the weighted balanced accuracy.

Usage

```
## S3 method for class 'factor'
baccuracy(actual, predicted, adjust = FALSE, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.baccuracy(actual, predicted, w, adjust = FALSE, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
baccuracy(x, adjust = FALSE, na.rm = TRUE, ...)
## Generic S3 method
baccuracy(
 adjust = FALSE,
 na.rm = TRUE
)
## Generic S3 method
weighted.baccuracy(
 W,
 adjust = FALSE,
 na.rm = TRUE
)
```

Arguments

actual A vector of <factor> with length n, and k levels predicted A vector of <factor> with length n, and k levels

baccuracy.factor 9

adjust	A logical value (default: FALSE). If TRUE the metric is adjusted for random chance $\frac{1}{k}$.
na.rm	A logical value (default: TRUE). If TRUE calculation of the metric is based on valid classes.
	Arguments passed into other methods
W	A <numeric>-vector of length n. NULL by default</numeric>
х	A confusion matrix created cmatrix()

Value

A numeric-vector of length 1

Definition

Let $\hat{\alpha} \in [0,1]$ be the proportion of correctly predicted classes. If adjust == false, the balanced accuracy of the classifier is calculated as,

$$\hat{\alpha} = \frac{\text{sensitivity} + \text{specificity}}{2}$$

otherwise,

$$\hat{\alpha} = \frac{\text{sensitivity} + \text{specificity}}{2} \frac{1}{k}$$

Where:

- k is the number of classes
- sensitivity is the overall sensitivity, and
- specificity is the overall specificity

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

10 baccuracy.factor

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(
    iris$Species == "virginica"
)
# 2) fit the logistic
# regression
model <- glm(
    formula = species_num ~ Sepal.Length + Sepal.Width,
    data = iris,
    family = binomial(
        link = "logit"
    )</pre>
```

banknote 11

```
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
 as.numeric(
   predict(model, type = "response") > 0.5
 ),
 levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
 levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 4) evaluate the
# model
cat(
  "Balanced accuracy", baccuracy(
            = actual,
   actual
   predicted = predicted
  "Balanced accuracy (weighted)", weighted.baccuracy(
   actual
             = actual,
   predicted = predicted,
              = iris$Petal.Length/mean(iris$Petal.Length)
 sep = "\n"
```

banknote

Banknote Authentication Dataset

Description

This dataset contains features extracted from the wavelet transform of banknote images, which are used to classify banknotes as authentic or inauthentic. The data originates from the UCI Machine Learning Repository.

Usage

```
data(banknote)
```

12 ccc.numeric

Format

A list with two components:

features A data frame with 4 variables: variance, skewness, curtosis, and entropy.

target A factor with levels "inauthentic" and "authentic" representing the banknote's authenticity.

Details

The data is provided as a list with two components:

features A data frame containing the following variables:

variance Variance of the wavelet transformed image.

skewness Skewness of the wavelet transformed image.

curtosis Curtosis of the wavelet transformed image.

entropy Entropy of the image.

target A factor indicating the authenticity of the banknote. The factor has two levels:

inauthentic Indicates the banknote is not genuine.

authentic Indicates the banknote is genuine.

Source

https://archive.ics.uci.edu/dataset/267/banknote+authentication

ccc.numeric

Concordance Correlation Coefficient

Description

A generic function for the concordance correlation coefficient. Use weighted.ccc() for the weighted concordance correlation coefficient.

Usage

ccc.numeric 13

```
...,
w,
correction = FALSE
)
```

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. A <logical> vector of length 1 (default: FALSE). If TRUE the variance and covariance will be adjusted with $\frac{1-n}{n}$... Arguments passed into other methods. A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A <numeric> vector of length 1.

Definition

Let $\rho_c \in [0,1]$ measure the agreement between y and v. The classifier agreement is calculated as,

$$\rho_c = \frac{2\rho\sigma_v\sigma_y}{\sigma_v^2 + \sigma_v^2 + (\mu_v - \mu_y)^2}$$

Where:

- ρ is the pearson correlation coefficient
- σ_y is the unbiased standard deviation of y
- σ_v is the unbiased standard deviation of v
- μ_y is the mean of y
- μ_{v} is the mean of v

If correction == TRUE each $\sigma_{i \in [y,v]}$ is adjusted by $\frac{1-n}{n}$

See Also

```
Other Regression: huberloss.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), mse.numeric(), pinball.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), specificity.factor(), zerooneloss.factor()
```

Examples

```
# 1) fit a linear
# regression
model <- lm(
 mpg ~ .,
  data = mtcars
# 1.1) define actual
# and predicted values
# to measure performance
actual
        <- mtcars$mpg
predicted <- fitted(model)</pre>
# 2) evaluate in-sample model
# performance
cat(
  "Concordance Correlation Coefficient", ccc(
              = actual,
   predicted = predicted,
   correction = FALSE
  "Concordance Correlation Coefficient (corrected)", ccc(
   actual
            = actual,
   predicted = predicted,
   correction = TRUE
  ),
  "Concordance Correlation Coefficient (weighhed)", weighted.ccc(
            = actual,
   predicted = predicted,
              = mtcars$mpg/mean(mtcars$mpg),
   correction = FALSE
  ),
  sep = "\n"
```

ckappa.factor

Cohen's \(\kappa\)-statistic

Description

A generic function for Cohen's κ -statistic. Use weighted.ckappa() for the weighted κ -statistic.

Usage

```
## S3 method for class 'factor'
ckappa(actual, predicted, beta = 0, ...)
## S3 method for class 'factor'
```

```
weighted.ckappa(actual, predicted, w, beta = 0, ...)
## S3 method for class 'cmatrix'
ckappa(x, beta = 0, ...)

ckappa(
    ...,
    beta = 0
)

weighted.ckappa(
    ...,
    w,
    beta = 0
)
```

Arguments

actual A vector of <factor> values of length n, and k levels.

A vector of <factor> values of length n, and k levels.

A <numeric> value of length 1 (default: 0). If $\beta \neq 0$ the off-diagonals of the confusion matrix are penalized with a factor of $(y_+ - y_{i,-})^{\beta}$.

Arguments passed into other methods

A <numeric>-vector of length n. NULL by default.

A confusion matrix created cmatrix().

Value

If micro is NULL (the default), a named <numeric>-vector of length k
If micro is TRUE or FALSE, a <numeric>-vector of length 1

Definition

Let $\kappa \in [0,1]$ be the inter-rater (intra-rater) reliability. The inter-rater (intra-rater) reliability is calculated as,

$$\kappa = \frac{\rho_p - \rho_e}{1 - \rho_e}$$

Where:

- ullet ρ_p is the empirical probability of agreement between predicted and actual values
- ρ_e is the expected probability of agreement under random chance

If $\beta \neq 0$ the off-diagonals in the confusion matrix is penalized before ρ is calculated. More formally,

$$\chi = X \circ Y^{\beta}$$

Where:

- X is the confusion matrix
- Y is the penalizing matrix and
- β is the penalizing factor

Creating < factor >

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(),
```

```
mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(),
pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(),
rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(),
rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(),
zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
  family = binomial(
   link = "logit"
  )
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
   predict(model, type = "response") > 0.5
  ),
  levels = c(1,0),
  labels = c("Virginica", "Others")
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
 levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 4) evaluate model performance with
# Cohens Kappa statistic
cat(
  "Kappa", ckappa(
   actual = actual,
   predicted = predicted
  "Kappa (penalized)", ckappa(
   actual = actual,
```

18 cmatrix.factor

```
predicted = predicted,
  beta = 2
),
"Kappa (weighted)", weighted.ckappa(
  actual = actual,
  predicted = predicted,
  w = iris$Petal.Length/mean(iris$Petal.Length)
),
  sep = "\n"
)
```

cmatrix.factor

Confusion Matrix

Description

The cmatrix()-function uses cross-classifying factors to build a confusion matrix of the counts at each combination of the factor levels. Each row of the matrix represents the actual factor levels, while each column represents the predicted factor levels.

Usage

```
## S3 method for class 'factor'
cmatrix(actual, predicted, ...)

## S3 method for class 'factor'
weighted.cmatrix(actual, predicted, w, ...)

## Generic S3 method
cmatrix(
    actual,
    predicted,
    ...
)

## Generic S3 method
weighted.cmatrix(
    actual,
    predicted,
    w,
    ...
)
```

Arguments

```
actual A <factor>-vector of length n, and k levels. predicted A <factor>-vector of length n, and k levels.
```

cmatrix.factor 19

... Arguments passed into other methods.

A <numeric>-vector of length n (default: NULL) If passed it will return a weighted confusion matrix.

Value

W

A named $k \times k < \text{matrix} >$

Dimensions

There is no robust defensive measure against mis-specifying the confusion matrix. If the arguments are correctly specified, the resulting confusion matrix is on the form:

	A (Predicted)	B (Predicted)
A (Actual)	Value	Value
B (Actual)	Value	Value

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

20 cmatrix.factor

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
        = iris,
  family = binomial(
    link = "logit"
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
    predict(model, type = "response") > 0.5
  ),
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
 levels = c(1,0),
  labels = c("Virginica", "Others")
)
```

```
# 4) summarise performance
# in a confusion matrix
# 4.1) unweighted matrix
confusion_matrix <- cmatrix(</pre>
  actual
           = actual,
  predicted = predicted
)
# 4.1.1) summarise matrix
summary(
  confusion_matrix
# 4.1.2) plot confusion
# matrix
plot(
  {\tt confusion\_matrix}
# 4.2) weighted matrix
confusion_matrix <- weighted.cmatrix(</pre>
  actual = actual,
  predicted = predicted,
           = iris$Petal.Length/mean(iris$Petal.Length)
)
# 4.2.1) summarise matrix
summary(
  {\tt confusion\_matrix}
)
# 4.2.1) plot confusion
# matrix
plot(
  {\tt confusion\_matrix}
```

dor.factor

Diagnostic Odds Ratio

Description

A generic function for the diagnostic odds ratio in classification tasks. Use weighted.dor() weighted diagnostic odds ratio.

Usage

```
## S3 method for class 'factor'
```

```
dor(actual, predicted, ...)
## S3 method for class 'factor'
weighted.dor(actual, predicted, w, ...)
## S3 method for class 'cmatrix'
dor(x, ...)
## Generic S3 method
dor(...)
## Generic S3 method
weighted.dor(
...,
w
)
```

Arguments

actual A vector of <factor> values of length n, and k levels.

A vector of <factor> values of length n, and k levels.

Arguments passed into other methods

A <numeric>-vector of length n. NULL by default.

A confusion matrix created cmatrix().

Value

A <numeric>-vector of length 1

Definition

Let $\hat{\alpha} \in [0, \infty]$ be the effectiveness of the classifier. The diagnostic odds ratio of the classifier is calculated as,

$$\hat{\alpha} = \frac{\text{\#TP\#TN}}{\text{\#FP\#FN}}$$

Where:

- #TP is the number of true positives
- #TN is the number of true negatives
- #FP is the number of false positives
- #FN is the number of false negatives

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), roc.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
  family = binomial(
   link = "logit"
  )
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
   predict(model, type = "response") > 0.5
 ),
 levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
 levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 4) evaluate model performance
# with Diagnostic Odds Ratio
cat("Diagnostic Odds Ratio", sep = "\n")
dor(
  actual
          = actual,
  predicted = predicted
cat("Diagnostic Odds Ratio (weighted)", sep = "\n")
weighted.dor(
  actual = actual,
  predicted = predicted,
           = iris$Petal.Length/mean(iris$Petal.Length)
)
```

entropy.matrix 25

entropy.matrix

Entropy

Description

The entropy() function calculates the **Entropy** of given probability distributions.

Usage

```
## S3 method for class 'matrix'
entropy(pk, dim = 0L, base = -1, ...)
## S3 method for class 'matrix'
relative.entropy(pk, qk, dim = 0L, base = -1, ...)
## S3 method for class 'matrix'
cross.entropy(pk, qk, dim = 0L, base = -1, ...)
## Generic S3 method
entropy(
рk,
dim = 0,
base = -1,
)
## Generic S3 method
relative.entropy(
pk,
qk,
dim = 0,
base = -1,
)
## Generic S3 method
cross.entropy(
рk,
qk,
dim = 0,
base = -1,
)
```

Arguments

pk

A $n \times k$ <numeric>-matrix of observed probabilities. The i-th row should sum to 1 (i.e., a valid probability distribution over the k classes). The first column

26 entropy.matrix

corresponds to the first factor level in actual, the second column to the second

factor level, and so on.

dim An <integer> value of length 1 (Default: 0). Defines the dimension along which

to calculate the entropy (0: total, 1: row-wise, 2: column-wise).

base A <numeric> value of length 1 (Default: -1). The logarithmic base to use. De-

fault value specifies natural logarithms.

... Arguments passed into other methods

qk A $n \times k$ <numeric>-matrix of predicted probabilities. The *i*-th row should sum

to 1 (i.e., a valid probability distribution over the k classes). The first column corresponds to the first factor level in actual, the second column to the second

factor level, and so on.

Value

A <numeric> value or vector:

• A single <numeric> value (length 1) if dim == 0.

• A <numeric> vector with length equal to the length of rows if dim == 1.

• A <numeric> vector with length equal to the length of columns if dim == 2.

Definition

Entropy:

$$H(pk) = -\sum_{i} pk_i \log(pk_i)$$

Cross Entropy:

$$H(pk, qk) = -\sum_{i} pk_i \log(qk_i)$$

Relative Entropy

$$D_{KL}(pk \parallel qk) = \sum_{i} pk_i \log \left(\frac{pk_i}{qk_i}\right)$$

See Also

Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()

Examples

```
# 1) Define actual
# and observed probabilities
# 1.1) actual probabilies
pk <- matrix(</pre>
  cbind(1/2, 1/2),
  ncol = 2
# 1.2) observed (estimated) probabilites
qk <- matrix(</pre>
  cbind(9/10, 1/10),
  ncol = 2
)
# 2) calculate
# Entropy
cat(
  "Entropy", entropy(
    pk
  "Relative Entropy", relative.entropy(
    pk,
    qk
  ),
  "Cross Entropy", cross.entropy(
    qk
  ),
  sep = "\n"
)
```

fbeta.factor

 F_β -score

Description

A generic function for the F_{β} -score. Use weighted fbeta() for the weighted F_{β} -score.

Usage

```
## S3 method for class 'factor'
fbeta(actual, predicted, beta = 1, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.fbeta(actual, predicted, w, beta = 1, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
```

Arguments

actual	A vector of $<$ factor $>$ values of length n , and k levels.
predicted	A vector of $<$ factor $>$ values of length n , and k levels.
beta	A <numeric> vector of length 1 (default: 1).</numeric>
micro	A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro average across all k classes, if FALSE it returns the macro average.</logical>
na.rm	A <logical> value of length 1 (default: TRUE). If TRUE, NA values are removed from the computation. This argument is only relevant when micro != NULL. When na.rm = TRUE, the computation corresponds to $sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))$. When na.rm = FALSE, the computation corresponds to $sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))$.</logical>
	Arguments passed into other methods
W	A <numeric>-vector of length n. NULL by default.</numeric>
х	A confusion matrix created cmatrix().

Value

```
If micro is NULL (the default), a named <numeric>-vector of length k
If micro is TRUE or FALSE, a <numeric>-vector of length 1
```

Definition

Let $\hat{F}_{\beta} \in [0,1]$ be the F_{β} score, which is a weighted harmonic mean of precision and recall. F_{β} score of the classifier is calculated as,

$$\hat{F}_{\beta} = (1 + \beta^2) \frac{\text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}}$$

Substituting Precision = $\frac{\#TP_k}{\#TP_k + \#FP_k}$ and Recall = $\frac{\#TP_k}{\#TP_k + \#FN_k}$ yields:

$$\hat{F}_{\beta} = (1 + \beta^2) \frac{\frac{\#TP_k}{\#TP_k + \#FP_k} \times \frac{\#TP_k}{\#TP_k + \#FN_k}}{\beta^2 \times \frac{\#TP_k}{\#TP_k + \#FP_k} + \frac{\#TP_k}{\#TP_k + \#FN_k}}$$

Where:

- $\#TP_k$ is the number of true positives,
- $\#FP_k$ is the number of false positives,
- $\#FN_k$ is the number of false negatives, and
- β is a non-negative real number that determines the relative importance of precision vs. recall
 in the score.

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
        = iris,
  family = binomial(
    link = "logit"
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
    predict(model, type = "response") > 0.5
  ),
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
 levels = c(1,0),
  labels = c("Virginica", "Others")
)
```

```
# 4) evaluate class-wise performance
# using F1-score
# 4.1) unweighted F1-score
fbeta(
 actual
            = actual,
 predicted = predicted,
            = 1
 beta
)
# 4.2) weighted F1-score
weighted.fbeta(
 actual
           = actual,
 predicted = predicted,
           = iris$Petal.Length/mean(iris$Petal.Length),
 beta
)
# 5) evaluate overall performance
# using micro-averaged F1-score
cat(
  "Micro-averaged F1-score", fbeta(
           = actual,
   actual
   predicted = predicted,
             = 1,
   beta
   micro
             = TRUE
  "Micro-averaged F1-score (weighted)", weighted.fbeta(
   actual
             = actual,
   predicted = predicted,
             = iris$Petal.Length/mean(iris$Petal.Length),
   beta
             = 1,
   micro
             = TRUE
 ),
 sep = "\n"
)
```

fdr.factor

false discovery rate

Description

A generic function for the False Discovery Rate. Use weighted.fdr() for the weighted False Discovery Rate.

Usage

```
## S3 method for class 'factor'
fdr(actual, predicted, micro = NULL, na.rm = TRUE, ...)
```

Arguments

actual	A vector of \prec factor \gt values of length n , and k levels.
predicted	A vector of \prec factor \gt values of length n , and k levels.
micro	A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro average across all k classes, if FALSE it returns the macro average.</logical>
na.rm	A <logical> value of length 1 (default: TRUE). If TRUE, NA values are removed from the computation. This argument is only relevant when micro != NULL. When na.rm = TRUE, the computation corresponds to $sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))$. When na.rm = FALSE, the computation corresponds to $sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))$.</logical>
	Arguments passed into other methods
W	A <numeric>-vector of length n. NULL by default.</numeric>
X	A confusion matrix created cmatrix().

Value

```
If micro is NULL (the default), a named <numeric>-vector of length k

If micro is TRUE or FALSE, a <numeric>-vector of length 1
```

Definition

Let $\hat{\alpha} \in [0,1]$ be the proportion of false positives among the preditced positives. The false discovery rate of the classifier is calculated as,

$$\hat{\alpha} = \frac{\#FP_k}{\#TP_k + \#FP_k}$$

Where:

- $\#TP_k$ is the number of true positives, and
- $\#FP_k$ is the number of false positives

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()
```

```
Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
  family = binomial(
    link = "logit"
  )
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
    predict(model, type = "response") > 0.5
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
  labels = c("Virginica", "Others")
# 4) evaluate class-wise performance
# using False Discovery Rate
# 4.1) unweighted False Discovery Rate
fdr(
  actual
          = actual,
```

fer.factor 35

```
predicted = predicted
# 4.2) weighted False Discovery Rate
weighted.fdr(
  actual
          = actual,
 predicted = predicted,
           = iris$Petal.Length/mean(iris$Petal.Length)
)
# 5) evaluate overall performance
# using micro-averaged False Discovery Rate
cat(
  "Micro-averaged False Discovery Rate", fdr(
            = actual,
   actual
   predicted = predicted,
           = TRUE
   micro
  ),
  "Micro-averaged False Discovery Rate (weighted)", weighted.fdr(
             = actual,
   predicted = predicted,
             = iris$Petal.Length/mean(iris$Petal.Length),
             = TRUE
   micro
  ),
  sep = "\n"
```

fer.factor

False Omission Rate

Description

A generic function for the false omission rate. Use weighted.fdr() for the weighted false omission rate.

Usage

```
## S3 method for class 'factor'
fer(actual, predicted, micro = NULL, na.rm = TRUE, ...)

## S3 method for class 'factor'
weighted.fer(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)

## S3 method for class 'cmatrix'
fer(x, micro = NULL, na.rm = TRUE, ...)

## Generic S3 method
fer(
```

36 fer.factor

```
micro = NULL,
na.rm = TRUE
)

## Generic S3 method
weighted.fer(
...,
w,
micro = NULL,
na.rm = TRUE
)
```

Arguments

actual A vector of <factor> values of length n, and k levels. predicted A vector of <factor> values of length n, and k levels. A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro micro average across all k classes, if FALSE it returns the macro average. A <logical> value of length 1 (default: TRUE). If TRUE, NA values are rena.rm moved from the computation. This argument is only relevant when micro!= NULL. When na.rm = TRUE, the computation corresponds to sum(c(1, 2, NA),na.rm = TRUE) / length(na.omit(c(1, 2, NA))). When na.rm = FALSE, the computation corresponds to sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA), na.rm = TRUE) / leng2, NA)). Arguments passed into other methods A <numeric>-vector of length n. NULL by default. W A confusion matrix created cmatrix(). Х

Value

If micro is NULL (the default), a named <numeric>-vector of length k

If micro is TRUE or FALSE, a <numeric>-vector of length 1

Definition

Let $\hat{\beta} \in [0, 1]$ be the proportion of false negatives among the predicted negatives. The false omission rate of the classifier is calculated as,

$$\hat{\beta} = \frac{\#FN_k}{\#TN_k + \#FN_k}$$

Where:

- $\#TN_k$ is the number of true negatives, and
- $\#FN_k$ is the number of false negatives.

fer.factor 37

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), roc.auc.matrix(), rrae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

38 fer.factor

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
 family = binomial(
   link = "logit"
  )
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
 as.numeric(
   predict(model, type = "response") > 0.5
 levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 4) evaluate class-wise performance
# using False Omission Rate
# 4.1) unweighted False Omission Rate
fer(
  actual
          = actual,
  predicted = predicted
# 4.2) weighted False Omission Rate
weighted.fer(
  actual = actual,
 predicted = predicted,
            = iris$Petal.Length/mean(iris$Petal.Length)
)
```

fmi.factor 39

```
# 5) evaluate overall performance
# using micro-averaged False Omission Rate
cat(
  "Micro-averaged False Omission Rate", fer(
   actual
             = actual,
   predicted = predicted,
             = TRUE
   micro
  ),
  "Micro-averaged False Omission Rate (weighted)", weighted.fer(
             = actual,
   actual
   predicted = predicted,
             = iris$Petal.Length/mean(iris$Petal.Length),
              = TRUE
   micro
  ),
  sep = "\n"
)
```

fmi.factor

Fowlkes-Mallows Index

Description

The fmi()-function computes the Fowlkes-Mallows Index (FMI), a measure of the similarity between two sets of clusterings, between two vectors of predicted and observed factor() values.

Usage

```
## S3 method for class 'factor'
fmi(actual, predicted, ...)
## S3 method for class 'cmatrix'
fmi(x, ...)
## Generic S3 method
fmi(...)
```

Arguments

actual A vector of <factor> with length n, and k levels predicted A vector of <factor> with length n, and k levels ... Arguments passed into other methods \mathbf{x} A confusion matrix created cmatrix()

Value

A <numeric>-vector of length 1

40 fmi.factor

Definition

The metric is calculated for each class k as follows,

$$\sqrt{\frac{\#TP_k}{\#TP_k + \#FP_k} \times \frac{\#TP_k}{\#TP_k + \#FN_k}}$$

Where $\#TP_k$, $\#FP_k$, and $\#FN_k$ represent the number of true positives, false positives, and false negatives for each class k, respectively.

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
  family = binomial(
    link = "logit"
  )
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
    predict(model, type = "response") > 0.5
 ),
 levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 4) evaluate model performance
# using Fowlkes Mallows Index
  "Fowlkes Mallows Index", fmi(
 actual = actual,
 predicted = predicted
  ),
  sep = "\n"
)
```

Description

A generic function for the False Positive Rate. Use weighted.fpr() for the weighted False Positive Rate

Other names:

Fallout

Usage

```
## S3 method for class 'factor'
fpr(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.fpr(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
fpr(x, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
fallout(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.fallout(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
fallout(x, micro = NULL, na.rm = TRUE, ...)
## Generic S3 method
fpr(
micro = NULL,
na.rm = TRUE
## Generic S3 method
fallout(
micro = NULL,
na.rm = TRUE
## Generic S3 method
weighted.fpr(
 . . . ,
W,
micro = NULL,
na.rm = TRUE
)
```

```
## Generic S3 method
weighted.fallout(
    ...,
    w,
    micro = NULL,
    na.rm = TRUE
)
```

Arguments

actual A vector of <factor> values of length n, and k levels. predicted A vector of <factor> values of length n, and k levels. micro A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro average across all k classes, if FALSE it returns the macro average. A <logical> value of length 1 (default: TRUE). If TRUE, NA values are rena.rm moved from the computation. This argument is only relevant when micro!= NULL. When na.rm = TRUE, the computation corresponds to sum(c(1, 2, NA),na.rm = TRUE) / length(na.omit(c(1, 2, NA))). When na.rm = FALSE, the computation corresponds to sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA), na.rm = TRUE) / leng2, NA)). Arguments passed into other methods A <numeric>-vector of length n. NULL by default. w A confusion matrix created cmatrix(). Х

Value

```
If micro is NULL (the default), a named <numeric>-vector of length k

If micro is TRUE or FALSE, a <numeric>-vector of length 1
```

Definition

Let $\hat{\gamma} \in [0, 1]$ be the proportion of false positives among the actual negatives. The false positive rate of the classifier is calculated as,

$$\hat{\gamma} = \frac{\#FP_k}{\#TN_k + \#FP_k}$$

Where:

- $\#TN_k$ is the number of true negatives, and
- $\#FP_k$ is the number of false positives.

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), roc.auc.matrix(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
 family = binomial(
   link = "logit"
  )
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
 as.numeric(
   predict(model, type = "response") > 0.5
 levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 4) evaluate class-wise performance
# using False Positive Rate
# 4.1) unweighted False Positive Rate
fpr(
  actual
          = actual,
  predicted = predicted
# 4.2) weighted False Positive Rate
weighted.fpr(
  actual = actual,
 predicted = predicted,
            = iris$Petal.Length/mean(iris$Petal.Length)
)
```

46 huberloss.numeric

```
# 5) evaluate overall performance
# using micro-averaged False Positive Rate
cat(
  "Micro-averaged False Positive Rate", fpr(
   actual
             = actual,
   predicted = predicted,
             = TRUE
   micro
  ),
  "Micro-averaged False Positive Rate (weighted)", weighted.fpr(
             = actual,
   predicted = predicted,
             = iris$Petal.Length/mean(iris$Petal.Length),
             = TRUE
   micro
  ),
  sep = "\n"
)
```

huberloss.numeric

Huber Loss

Description

The huberloss()-function computes the simple and weighted huber loss between the predicted and observed <numeric> vectors. The weighted.huberloss() function computes the weighted Huber Loss.

Usage

```
## S3 method for class 'numeric'
huberloss(actual, predicted, delta = 1, ...)
## S3 method for class 'numeric'
weighted.huberloss(actual, predicted, w, delta = 1, ...)
## Generic S3 method
huberloss(
 actual,
predicted,
delta = 1,
## Generic S3 method
weighted.huberloss(
 actual,
predicted,
W,
delta = 1,
)
```

huberloss.numeric 47

Arguments

actual	A $<$ numeric $>$ -vector of length n . The observed (continuous) response variable.
predicted	A $<$ numeric $>$ -vector of length n . The estimated (continuous) response variable.
delta	A <numeric>-vector of length 1 (default: 1). The threshold value for switch between functions (see calculation).</numeric>
	Arguments passed into other methods.
W	A $<$ numeric $>$ -vector of length n . The weight assigned to each observation in the data.

Value

A <numeric> vector of length 1.

Definition

The metric is calculated as follows,

$$\frac{1}{2}(y-v)^2 for |y-v| \le \delta$$

and

$$\delta |y-v| - \frac{1}{2}\delta^2 \ for \ \text{otherwise}$$

where y and v are the actual and predicted values respectively. If w is not NULL, then all values are aggregated using the weights.

See Also

```
pinball.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rrmse.numeric(),
rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(),
ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(),
fdr.factor(), fer.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mae.numeric(),
mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(),
pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(),
rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(),
rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(),
zerooneloss.factor()
```

Other Regression: ccc.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), mse.numeric(),

```
# 1) fit a linear
# regression
model <- lm(
   mpg ~ .,
   data = mtcars</pre>
```

```
)
# 1.1) define actual
# and predicted values
# to measure performance
actual
         <- mtcars$mpg
predicted <- fitted(model)</pre>
# 2) calculate the metric
# with delta 0.5
huberloss(
  actual = actual,
  predicted = predicted,
  delta = 0.5
# 3) caclulate weighted
# metric using arbitrary weights
w <- rbeta(
  n = 1e3,
  shape1 = 10,
  shape2 = 2
)
huberloss(
  actual = actual,
  predicted = predicted,
  delta = 0.5,
        = w
)
```

jaccard.factor

Jaccard Index

Description

The <code>jaccard()</code>-function computes the <code>Jaccard Index</code>, also known as the Intersection over Union, between two vectors of predicted and observed <code>factor()</code> values. The <code>weighted.jaccard()</code> function computes the weighted <code>Jaccard Index</code>.

Usage

```
## S3 method for class 'factor'
jaccard(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.jaccard(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
```

```
## S3 method for class 'cmatrix'
jaccard(x, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
csi(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.csi(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
csi(x, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
tscore(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.tscore(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
tscore(x, micro = NULL, na.rm = TRUE, ...)
## Generic S3 method
jaccard(
 ...,
micro = NULL,
na.rm = TRUE
## Generic S3 method
csi(
 . . . ,
micro = NULL,
na.rm = TRUE
## Generic S3 method
tscore(
micro = NULL,
na.rm = TRUE
## Generic S3 method
weighted.jaccard(
 ...,
micro = NULL,
 na.rm = TRUE
```

```
## Generic S3 method
weighted.csi(
    ...,
    w,
    micro = NULL,
    na.rm = TRUE
)

## Generic S3 method
weighted.tscore(
    ...,
    w,
    micro = NULL,
    na.rm = TRUE
)
```

Arguments

actual predicted	A vector of \prec factor \gt values of length n , and k levels. A vector of \prec factor \gt values of length n , and k levels.
micro	A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro average across all k classes, if FALSE it returns the macro average.</logical>
na.rm	A <logical> value of length 1 (default: TRUE). If TRUE, NA values are removed from the computation. This argument is only relevant when micro != NULL. When na.rm = TRUE, the computation corresponds to $sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))$. When na.rm = FALSE, the computation corresponds to $sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))$.</logical>
	Arguments passed into other methods
W	A <numeric>-vector of length n. NULL by default.</numeric>
x	A confusion matrix created cmatrix().

Value

```
If micro is NULL (the default), a named <numeric>-vector of length k
If micro is TRUE or FALSE, a <numeric>-vector of length 1
```

Definition

The metric is calculated for each class k as follows,

$$\frac{\#TP_k}{\#TP_k + \#FP_k + \#FN_k}$$

Where $\#TP_k$, $\#FP_k$, and $\#FN_k$ represent the number of true positives, false positives, and false negatives for each class k, respectively.

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
 family = binomial(
   link = "logit"
  )
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
 as.numeric(
   predict(model, type = "response") > 0.5
 levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 4) evaluate class-wise performance
# using Jaccard Index
# 4.1) unweighted Jaccard Index
jaccard(
  actual
            = actual,
  predicted = predicted
# 4.2) weighted Jaccard Index
weighted.jaccard(
  actual = actual,
  predicted = predicted,
            = iris$Petal.Length/mean(iris$Petal.Length)
)
```

logloss.factor 53

```
# 5) evaluate overall performance
# using micro-averaged Jaccard Index
cat(
  "Micro-averaged Jaccard Index", jaccard(
   actual
             = actual,
   predicted = predicted,
             = TRUE
   micro
  ),
  "Micro-averaged Jaccard Index (weighted)", weighted.jaccard(
             = actual,
   predicted = predicted,
             = iris$Petal.Length/mean(iris$Petal.Length),
              = TRUE
   micro
  ),
  sep = "\n"
)
```

logloss.factor

Log Loss

Description

The logloss() function computes the **Log Loss** between observed classes (as a <factor>) and their predicted probability distributions (a <numeric> matrix). The weighted.logloss() function is the weighted version, applying observation-specific weights.

Usage

```
## S3 method for class 'factor'
logloss(actual, response, normalize = TRUE, ...)

## S3 method for class 'factor'
weighted.logloss(actual, response, w, normalize = TRUE, ...)

## S3 method for class 'integer'
logloss(actual, response, normalize = TRUE, ...)

## S3 method for class 'integer'
weighted.logloss(actual, response, w, normalize = TRUE, ...)

## Generic S3 method
logloss(
actual,
response,
normalize = TRUE,
...
)
```

54 logloss.factor

```
## Generic S3 method
weighted.logloss(
   actual,
   response,
   w,
   normalize = TRUE,
   ...
)
```

Arguments

actual A vector of <factor> with length n, and k levels

response A $n \times k$ <numeric>-matrix of predicted probabilities. The i-th row should sum to 1 (i.e., a valid probability distribution over the k classes). The first column corresponds to the first factor level in actual, the second column to the second factor level, and so on.

normalize A <logical>-value (default: TRUE). If TRUE, the mean cross-entropy across all observations is returned; otherwise, the sum of cross-entropies is returned.

Arguments passed into other methods

A <numeric>-vector of length n. NULL by default

Value

A <numeric>-vector of length 1

Definition

$$H(p, response) = -\sum_{i} \sum_{j} y_{ij} \log_2(response_{ij})$$

where:

- y_{ij} is the actual-values, where $y_{ij} = 1$ if the i-th sample belongs to class j, and 0 otherwise.
- $response_{ij}$ is the estimated probability for the i-th sample belonging to class j.

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)
## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
```

logloss.factor 55

```
)
#> [1] B A B B A C B C C A
#> Levels: A B C

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does
not predict any B's. The predicted vector of factor() values would be defined as follows:

## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) Recode the iris data set to a binary classification problem
# Here, the positive class ("Virginica") is coded as 1,
# and the rest ("Others") is coded as 0.
iris$species_num <- as.numeric(iris$Species == "virginica")
# 2) Fit a logistic regression model predicting species_num from Sepal.Length & Sepal.Width model <- glm(
   formula = species_num ~ Sepal.Length + Sepal.Width,
   data = iris,
   family = binomial(link = "logit")</pre>
```

56 mae.numeric

```
)
# 3) Generate predicted classes: "Virginica" vs. "Others"
predicted <- factor(</pre>
  as.numeric(predict(model, type = "response") > 0.5),
  levels = c(1, 0),
  labels = c("Virginica", "Others")
# 3.1) Generate actual classes
actual <- factor(</pre>
        = iris$species_num,
  levels = c(1, 0),
  labels = c("Virginica", "Others")
)
# For Log Loss, we need predicted probabilities for each class.
# Since it's a binary model, we create a 2-column matrix:
# 1st column = P("Virginica")
    2nd column = P("Others") = 1 - P("Virginica")
predicted_probs <- predict(model, type = "response")</pre>
response_matrix <- cbind(predicted_probs, 1 - predicted_probs)</pre>
# 4) Evaluate unweighted Log Loss
     'logloss' takes (actual, response_matrix, normalize=TRUE/FALSE).
     The factor 'actual' must have the positive class (Virginica) as its first level.
unweighted_LogLoss <- logloss(</pre>
                                 # factor
  actual
          = actual,
  response = response_matrix, # numeric matrix of probabilities
  normalize = TRUE
                                 # normalize = TRUE
)
# 5) Evaluate weighted Log Loss
     We introduce a weight vector, for example:
weights <- iris$Petal.Length / mean(iris$Petal.Length)</pre>
weighted_LogLoss <- weighted.logloss(</pre>
  actual
          = actual,
  response = response_matrix,
            = weights,
  normalize = TRUE
# 6) Print Results
cat(
  "Unweighted Log Loss:", unweighted_LogLoss,
  "Weighted Log Loss:", weighted_LogLoss,
  sep = "\n"
)
```

mae.numeric 57

Description

The mae()-function computes the mean absolute error between the observed and predicted <numeric> vectors. The weighted.mae() function computes the weighted mean absolute error.

Usage

```
## S3 method for class 'numeric'
mae(actual, predicted, ...)

## S3 method for class 'numeric'
weighted.mae(actual, predicted, w, ...)

## Generic S3 method
mae(
    actual,
    predicted,
    ...
)

## Generic S3 method
weighted.mae(
    actual,
    predicted,
    w,
    ...
)
```

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. Arguments passed into other methods. A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A <numeric> vector of length 1.

Definition

The metric is calulated as follows,

$$\frac{\sum_{i=1}^{n}|y_{i}-v_{i}|}{n}$$

58 mape.numeric

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mape.numeric(), mpe.numeric(), mse.numeric(), pinball.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) fit a linear
# regression
model <- lm(</pre>
  mpg \sim .,
  data = mtcars
# 1.1) define actual
# and predicted values
# to measure performance
actual
         <- mtcars$mpg
predicted <- fitted(model)</pre>
# 2) evaluate in-sample model
# performance using Mean Absolute Error (MAE)
cat(
  "Mean Absolute Error", mae(
    actual = actual,
    predicted = predicted,
  ),
  "Mean Absolute Error (weighted)", weighted.mae(
    actual
            = actual,
    predicted = predicted,
              = mtcars$mpg/mean(mtcars$mpg)
  ),
  sep = "\n"
```

mape.numeric 59

Description

The mape()-function computes the mean absolute percentage error between the observed and predicted <numeric> vectors. The weighted.mape() function computes the weighted mean absolute percentage error.

Usage

```
## S3 method for class 'numeric'
mape(actual, predicted, ...)

## S3 method for class 'numeric'
weighted.mape(actual, predicted, w, ...)

## Generic S3 method
mape(
    actual,
    predicted,
    ...
)

## Generic S3 method
weighted.mape(
    actual,
    predicted,
    w,
    ...
)
```

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. Arguments passed into other methods.

W A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A < numeric > vector of length 1.

Definition

The metric is calculated as,

$$\frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - v_i|}{|y_i|}$$

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mpe.numeric(), mse.numeric(), pinball.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) fit a linear
# regression
model <- lm(</pre>
  mpg \sim .,
  data = mtcars
# 1.1) define actual
# and predicted values
# to measure performance
         <- mtcars$mpg
predicted <- fitted(model)</pre>
# 2) evaluate in-sample model
# performance using Mean Absolute Percentage Error (MAPE)
cat(
  "Mean Absolute Percentage Error", mape(
    actual = actual,
    predicted = predicted,
  "Mean Absolute Percentage Error (weighted)", weighted.mape(
    actual
            = actual,
    predicted = predicted,
              = mtcars$mpg/mean(mtcars$mpg)
  ),
  sep = "\n"
```

Description

The mcc()-function computes the Matthews Correlation Coefficient (MCC), also known as the ϕ -coefficient, between two vectors of predicted and observed factor() values. The weighted.mcc() function computes the weighted Matthews Correlation Coefficient.

Usage

```
## S3 method for class 'factor'
mcc(actual, predicted, ...)
## S3 method for class 'factor'
weighted.mcc(actual, predicted, w, ...)
## S3 method for class 'cmatrix'
mcc(x, ...)
## S3 method for class 'factor'
phi(actual, predicted, ...)
## S3 method for class 'factor'
weighted.phi(actual, predicted, w, ...)
## S3 method for class 'cmatrix'
phi(x, ...)
## Generic S3 method
mcc(...)
## Generic S3 method
weighted.mcc(
 . . . ,
W
)
## Generic S3 method
phi(...)
## Generic S3 method
weighted.phi(
)
```

Arguments

actual A vector of <factor> with length n, and k levels predicted A vector of <factor> with length n, and k levels ... Arguments passed into other methods

w A <numeric>-vector of length n. NULL by default

x A confusion matrix created cmatrix()

Value

A <numeric>-vector of length 1

Definition

The metric is calculated as follows,

$$\frac{\#TP \times \#TN - \#FP \times \#FN}{\sqrt{(\#TP + \#FP)(\#TP + \#FN)(\#TN + \#FP)(\#TN + \#FN)}}$$

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
        = iris,
  family = binomial(
    link = "logit"
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
    predict(model, type = "response") > 0.5
  ),
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
 levels = c(1,0),
  labels = c("Virginica", "Others")
)
```

mpe.numeric

```
# 4) evaluate performance
# using Matthews Correlation Coefficient
cat(
   "Matthews Correlation Coefficient", mcc(
      actual = actual,
      predicted = predicted
),
   "Matthews Correlation Coefficient (weighted)", weighted.mcc(
      actual = actual,
      predicted = predicted,
      w = iris$Petal.Length/mean(iris$Petal.Length)
),
   sep = "\n"
)
```

mpe.numeric

Mean Percentage Error

Description

The mpe()-function computes the mean percentage error between the observed and predicted <numeric> vectors. The weighted.mpe() function computes the weighted mean percentage error.

Usage

```
## S3 method for class 'numeric'
mpe(actual, predicted, ...)

## S3 method for class 'numeric'
weighted.mpe(actual, predicted, w, ...)

## Generic S3 method
mpe(
    actual,
    predicted,
    ...
)

## Generic S3 method
weighted.mpe(
    actual,
    predicted,
    w,
    ...
)
```

mpe.numeric 65

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. Arguments passed into other methods.

W A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A <numeric> vector of length 1.

Definition

The metric is calculated as,

$$\frac{1}{n} \sum_{i=1}^{n} \frac{y_i - v_i}{y_i}$$

Where y_i and v_i are the actual and predicted values respectively.

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mape.numeric(), mse.numeric(), pinball.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) fit a linear
# regression
model <- lm(
   mpg ~ .,
   data = mtcars
)
# 1.1) define actual
# and predicted values
# to measure performance
actual <- mtcars$mpg
predicted <- fitted(model)</pre>
```

66 mse.numeric

```
# 2) evaluate in-sample model
# performance using Mean Percentage Error (MPE)
cat(
   "Mean Percentage Error", mpe(
      actual = actual,
      predicted = predicted,
   ),
   "Mean Percentage Error (weighted)", weighted.mpe(
      actual = actual,
      predicted = predicted,
      w = mtcars$mpg/mean(mtcars$mpg)
   ),
   sep = "\n"
)
```

mse.numeric

Mean Squared Error

Description

The mse()-function computes the mean squared error between the observed and predicted <numeric> vectors. The weighted.mse() function computes the weighted mean squared error.

Usage

```
## S3 method for class 'numeric'
mse(actual, predicted, ...)

## S3 method for class 'numeric'
weighted.mse(actual, predicted, w, ...)

## Generic S3 method
mse(
    actual,
    predicted,
    ...
)

## Generic S3 method
weighted.mse(
    actual,
    predicted,
    w,
    ...
)
```

mse.numeric 67

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. Arguments passed into other methods.

W A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A <numeric> vector of length 1.

Definition

The metric is calculated as,

$$\frac{1}{n}\sum_{i}^{n}(y_i-v_i)^2$$

Where y_i and v_i are the actual and predicted values respectively.

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), pinball.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) fit a linear
# regression
model <- lm(
   mpg ~ .,
   data = mtcars
)
# 1.1) define actual
# and predicted values
# to measure performance
actual <- mtcars$mpg
predicted <- fitted(model)</pre>
```

68 nlr.factor

```
# 2) evaluate in-sample model
# performance using Mean Squared Error (MSE)
cat(
   "Mean Squared Error", mse(
    actual = actual,
    predicted = predicted,
   ),
   "Mean Squared Error (weighted)", weighted.mse(
    actual = actual,
    predicted = predicted,
    w = mtcars$mpg/mean(mtcars$mpg)
   ),
   sep = "\n"
)
```

nlr.factor

Negative Likelihood Ratio

Description

A generic function for the negative likelihood ratio in classification tasks. Use weighted.nlr() weighted negative likelihood ratio.

Usage

```
## S3 method for class 'factor'
nlr(actual, predicted, ...)

## S3 method for class 'factor'
weighted.nlr(actual, predicted, w, ...)

## S3 method for class 'cmatrix'
nlr(x, ...)

## Generic S3 method
nlr(...)

## Generic S3 method
weighted.nlr(
...,
w
)
```

Arguments

actual A vector of <factor> values of length n, and k levels. predicted A vector of <factor> values of length n, and k levels.

nlr.factor 69

... Arguments passed into other methods

w A <numeric>-vector of length n. NULL by default.

x A confusion matrix created cmatrix().

Value

If micro is NULL (the default), a named <numeric>-vector of length k
If micro is TRUE or FALSE, a <numeric>-vector of length 1

Definition

Let $\hat{\alpha} \in [0, \infty]$ be the likelihood of a negative outcome. The negative likelihood ratio of the classifier is calculated as,

$$\hat{\alpha} = \frac{1 - \frac{\#TP}{\#TP + \#FN}}{\frac{\#TN}{\#TN + \#FP}}$$

Where:

- $\frac{\#TP}{\#TP + \#FN}$ is the sensitivity, or true positive rate
- $\frac{\#TN}{\#TN + \#FP}$ is the specificity, or true negative rate

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)
## predicted
factor(
```

70 nlr.factor

```
x = sample(x = c(1, 3), size = 10, replace = TRUE),
levels = c(1, 2, 3),
labels = c("A", "B", "C")
)
#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k=3, determined indirectly by the levels argument.

See Also

```
The plr()-function for the Positive Likehood Ratio (LR+)
```

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
 iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
 formula = species_num ~ Sepal.Length + Sepal.Width,
 data = iris,
 family = binomial(
   link = "logit"
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
 as.numeric(
    predict(model, type = "response") > 0.5
```

npv.factor 71

```
),
  levels = c(1,0),
  labels = c("Virginica", "Others")
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
  labels = c("Virginica", "Others")
# 4) evaluate model performance
# with class-wise negative likelihood ratios
cat("Negative Likelihood Ratio", sep = "\n")
nlr(
  actual
            = actual,
  predicted = predicted
cat("Negative Likelihood Ratio (weighted)", sep = "\n")
weighted.nlr(
  actual
          = actual,
  predicted = predicted,
            = iris$Petal.Length/mean(iris$Petal.Length)
)
```

npv.factor

Negative Predictive Value

Description

The npv()-function computes the negative predictive value, also known as the True Negative Predictive Value, between two vectors of predicted and observed factor() values. The weighted.npv() function computes the weighted negative predictive value.

Usage

```
## S3 method for class 'factor'
npv(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.npv(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
npv(x, micro = NULL, na.rm = TRUE, ...)
npv(...)
```

72 npv.factor

```
weighted.npv(...)
```

Arguments

actual A vector of <factor> values of length n, and k levels. predicted A vector of <factor> values of length n, and k levels. A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro micro average across all k classes, if FALSE it returns the macro average. na.rm A < logical > value of length 1 (default: TRUE). If TRUE, NA values are removed from the computation. This argument is only relevant when micro!= NULL. When na.rm = TRUE, the computation corresponds to sum(c(1, 2, NA),na.rm = TRUE) / length(na.omit(c(1, 2, NA))). When na.rm = FALSE, the computation corresponds to sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA), na.rm = TRUE) / leng2, NA)). Arguments passed into other methods A < numeric >- vector of length n. NULL by default.

W

A confusion matrix created cmatrix(). Χ

Value

If micro is NULL (the default), a named < numeric >-vector of length k If micro is TRUE or FALSE, a < numeric >- vector of length 1

Definition

The metric is calculated for each class k as follows,

$$\frac{\#TN_k}{\#TN_k + \#FN_k}$$

Where $\#TN_k$ and $\#FN_k$ are the number of true negatives and false negatives, respectively, for each class k.

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)
## actual
factor(
  x = sample(x = 1:3, size = 10, replace = TRUE),
 levels = c(1, 2, 3),
  labels = c("A", "B", "C")
```

npv.factor 73

```
"> [1] B A B B A C B C C A

#> Levels: A B C

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

## set seed
set.seed(1903)

## predicted
```

predicted
factor(
 x = sample(x = c(1, 3), size = 10, replace = TRUE),
 levels = c(1, 2, 3),
 labels = c("A", "B", "C")
)
#> [1] C A C C C C C C A C
#> Levels: A B C

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rsg.numeric(), rmsle.numeric(), specificity.factor(), zerooneloss.factor()
```

Examples

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(
   iris$Species == "virginica"
)
# 2) fit the logistic
# regression
model <- glm(</pre>
```

74 npv.factor

```
formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
  family = binomial(
   link = "logit"
  )
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
   predict(model, type = "response") > 0.5
  ),
  levels = c(1,0),
 labels = c("Virginica", "Others")
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
 levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 4) evaluate class-wise performance
# using Negative Predictive Value
# 4.1) unweighted Negative Predictive Value
npv(
  actual
          = actual,
  predicted = predicted
# 4.2) weighted Negative Predictive Value
weighted.npv(
  actual = actual,
  predicted = predicted,
          = iris$Petal.Length/mean(iris$Petal.Length)
)
# 5) evaluate overall performance
# using micro-averaged Negative Predictive Value
cat(
  "Micro-averaged Negative Predictive Value", npv(
   actual
            = actual,
   predicted = predicted,
   micro
             = TRUE
  "Micro-averaged Negative Predictive Value (weighted)", weighted.npv(
   actual = actual,
   predicted = predicted,
             = iris$Petal.Length/mean(iris$Petal.Length),
```

obesity 75

```
micro = TRUE
),
sep = "\n"
)
```

obesity

Obesity Levels Dataset

Description

This dataset is used to estimate obesity levels based on eating habits and physical condition. The data originates from the UCI Machine Learning Repository and has been preprocessed to include both predictors and a target variable.

Usage

```
data(obesity)
```

Format

A list with two components:

features A data frame containing various predictors related to eating habits, physical condition, and lifestyle.

target A list with two elements: regression (weight in kilograms) and class (obesity level classification).

Details

The dataset is provided as a list with two components:

features A data frame containing various predictors related to lifestyle, eating habits, and physical condition. The variables include:

age The age of the individual in years.

height The height of the individual in meters.

family_history_with_overweight Binary variable indicating whether the individual has a family history of overweight (1 = yes, 0 = no).

favc Binary variable indicating whether the individual frequently consumes high-calorie foods (1 = yes, 0 = no).

fcvc The frequency of consumption of vegetables in meals.

ncp The number of main meals consumed per day.

caec Categorical variable indicating the frequency of consumption of food between meals. Typical levels include "no", "sometimes", "frequently", and "always".

smoke Binary variable indicating whether the individual smokes (1 = yes, 0 = no).

ch2o Daily water consumption (typically in liters).

76 openmp.on

scc Binary variable indicating whether the individual monitors calorie consumption (1 = yes, 0 = no).

faf The frequency of physical activity.

tue The time spent using electronic devices (e.g., screen time in hours).

calc Categorical variable indicating the frequency of alcohol consumption. Typical levels include "no", "sometimes", "frequently", and "always".

male Binary variable indicating the gender of the individual (1 = male, 0 = female).

target A list containing two elements:

regression A numeric vector representing the weight of the individual (used as the regression target).

class A factor indicating the obesity level classification. The levels are derived from the original nobeyesdad variable in the dataset.

Source

https://archive.ics.uci.edu/dataset/544/estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition

openmp.on

Use OpenMP

Description

This function allows you to enable or disable the use of OpenMP for parallelizing computations.

Usage

```
## enable OpenMP
openmp.on()
## disable OpenMP
openmp.off()
## set number of threads
openmp.threads(threads)
```

Arguments

threads

A positive <integer>-value (Default: None). If threads is missing, the openmp. threads() returns the number of available threads. If NULL all available threads will be used.

Value

```
If OpenMP is unavailable, the function returns NULL. If OpenMP is unavailable, the function returns NULL. If OpenMP is unavailable, the function returns NULL.
```

pinball.numeric 77

Examples

```
## Not run:
    ## enable OpenMP
    SLmetrics::openmp.on()

## disable OpenMP
    SLmetrics::openmp.off()

## available threads
    SLmetrics::openmp.threads()

## set number of threads
    SLmetrics::openmp.threads(2)

## End(Not run)
```

pinball.numeric

Pinball Loss

Description

The pinball()-function computes the pinball loss between the observed and predicted <numeric> vectors. The weighted.pinball() function computes the weighted Pinball Loss.

```
## S3 method for class 'numeric'
pinball(actual, predicted, alpha = 0.5, deviance = FALSE, ...)
## S3 method for class 'numeric'
weighted.pinball(actual, predicted, w, alpha = 0.5, deviance = FALSE, ...)
## Generic S3 method
pinball(
actual,
predicted,
 alpha = 0.5,
 deviance = FALSE,
)
## Generic S3 method
weighted.pinball(
actual,
predicted,
W,
```

78 pinball.numeric

```
alpha = 0.5,
deviance = FALSE,
...
)
```

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. A <numeric>-value of length 1 (default: 0.5). The slope of the pinball loss function. A <logical>-value of length 1 (default: FALSE). If TRUE the function returns the D^2 loss. Arguments passed into other methods. A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A <numeric> vector of length 1.

Definition

The metric is calculated as,

PinballLoss_{unweighted} =
$$\frac{1}{n} \sum_{i=1}^{n} \left[\alpha \cdot \max(0, y_i - \hat{y}_i) - (1 - \alpha) \cdot \max(0, \hat{y}_i - y_i) \right]$$

where y_i is the actual value, \hat{y}_i is the predicted value and α is the quantile level.

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rmsle.numeric(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), nsse.numeric(), nlr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rsg.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrsse.numeric(), rsg.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

plr.factor 79

Examples

```
# 1) fit a linear
# regression
model <- lm(
 mpg ~ .,
  data = mtcars
# 1.1) define actual
# and predicted values
# to measure performance
actual
         <- mtcars$mpg
predicted <- fitted(model)</pre>
# 2) evaluate in-sample model
# performance using Pinball Loss
  "Pinball Loss", pinball(
   actual
           = actual,
   predicted = predicted,
  "Pinball Loss (weighted)", weighted.pinball(
   actual = actual,
   predicted = predicted,
             = mtcars$mpg/mean(mtcars$mpg)
  ),
  sep = "\n"
)
```

plr.factor

Positive Likelihood Ratio

Description

A generic function for the positive likelihood ratio in classification tasks. Use weighted.plr() weighted positive likelihood ratio.

```
## S3 method for class 'factor'
plr(actual, predicted, ...)

## S3 method for class 'factor'
weighted.plr(actual, predicted, w, ...)

## S3 method for class 'cmatrix'
plr(x, ...)

## Generic S3 method
```

80 plr.factor

```
plr(...)
## Generic S3 method
weighted.plr(
    ...,
    w
)
```

Arguments

actual A vector of <factor> values of length n, and k levels.

A vector of <factor> values of length n, and k levels.

Arguments passed into other methods

A <numeric>-vector of length n. NULL by default.

A confusion matrix created cmatrix().

Value

If micro is NULL (the default), a named <numeric>-vector of length k

If micro is TRUE or FALSE, a <numeric>-vector of length 1

Definition

Let $\hat{\alpha} \in [0, \infty]$ be the likelihood of a positive outcome. The positive likelihood ratio of the classifier is calculated as,

$$\hat{\alpha} = \frac{\frac{\#TP}{\#TP + \#FN}}{1 - \frac{\#TN}{\#TN + \#FP}}$$

Where:

- $\frac{\#TP}{\#TP + \#FN}$ is the sensitivity, or true positive rate
- $\frac{\#TN}{\#TN + \#FP}$ is the specificity, or true negative rate

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)
## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
```

plr.factor 81

```
#> [1] B A B B A C B C C A
   #> Levels: A B C
   Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does
   not predict any B's. The predicted vector of factor() values would be defined as follows:
    ## set seed
   set.seed(1903)
   ## predicted
   factor(
      x = sample(x = c(1, 3), size = 10, replace = TRUE),
     levels = c(1, 2, 3),
      labels = c("A", "B", "C")
   #> [1] C A C C C C C A C
   #> Levels: A B C
   In both cases, k = 3, determined indirectly by the levels argument.
See Also
   The nlr()-function for the Negative Likehood Ratio (LR-)
   Other Classification: ROC. factor(), accuracy. factor(), baccuracy. factor(), ckappa. factor(),
    cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(),
    fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(),
    npv.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(),
    roc.auc.matrix(), specificity.factor(), zerooneloss.factor()
    Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(),
    ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(),
    fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(),
   mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(),
    npv.factor(),pinball.numeric(),pr.auc.matrix(),prROC.factor(),precision.factor(),
    rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(),
    rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(),
    zerooneloss.factor()
Examples
   # 1) recode Iris
   # to binary classification
    # problem
   iris$species_num <- as.numeric(</pre>
     iris$Species == "virginica"
    # 2) fit the logistic
    # regression
```

```
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
        = iris,
  data
  family = binomial(
   link = "logit"
  )
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
   predict(model, type = "response") > 0.5
  ),
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 4) evaluate model performance
# with class-wise positive likelihood ratios
cat("Positive Likelihood Ratio", sep = "\n")
plr(
  actual
            = actual,
  predicted = predicted
)
cat("Positive Likelihood Ratio (weighted)", sep = "\n")
weighted.plr(
  actual
          = actual,
  predicted = predicted,
           = iris$Petal.Length/mean(iris$Petal.Length)
)
```

pr.auc.matrix

Area under the Precision-Recall Curve

Description

A generic function for the area under the Precision-Recall Curve. Use weighted.pr.auc() for the weighted area under the Precision-Recall Curve.

Usage

```
## S3 method for class 'matrix'
pr.auc(actual, response, micro = NULL, method = 0L, ...)
## S3 method for class 'matrix'
weighted.pr.auc(actual, response, w, micro = NULL, method = 0L, ...)
## Generic S3 method
pr.auc(
actual,
 response,
 micro = NULL,
 method = 0,
)
## Generic S3 method
weighted.pr.auc(
 actual,
 response,
 W,
 micro = NULL,
 method = 0,
)
```

Arguments

actual	A vector of $<$ factor $>$ values of length n , and k levels.
response	A $n \times k$ <numeric>-matrix. The estimated response probabilities for each class k.</numeric>
micro	A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro average across all k classes, if FALSE it returns the macro average.</logical>
method	A <numeric> value (default: 0). Defines the underlying method of calculating the area under the curve. If 0 it is calculated using the trapezoid-method, if 1 it is calculated using the step-method.</numeric>
• • •	Arguments passed into other methods.
W	A <numeric>-vector of length n. NULL by default.</numeric>

Value

A <numeric> vector of length 1

Definition

Trapezoidal rule

The **trapezoidal rule** approximates the integral of a function f(x) between x = a and x = b using trapezoids formed between consecutive points. If we have points x_0, x_1, \ldots, x_n (with $a = x_0 < x_1 < \cdots < x_n = b$) and corresponding function values $f(x_0), f(x_1), \ldots, f(x_n)$, the area under the curve A_T is approximated by:

$$A_T \approx \sum_{k=1}^{n} \frac{f(x_{k-1}) + f(x_k)}{2} [x_k - x_{k-1}].$$

Step-function method

The **step-function** (**rectangular**) **method** uses the value of the function at one endpoint of each subinterval to form rectangles. With the same partition x_0, x_1, \ldots, x_n , the rectangular approximation A_S can be written as:

$$A_S \approx \sum_{k=1}^{n} f(x_{k-1}) [x_k - x_{k-1}].$$

See Also

Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()

Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()

Examples

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
 family = binomial(
    link = "logit"
  )
)
# 3) generate predicted
# classes
response <- predict(model, type = "response")</pre>
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
 levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 4) generate precision-recall
# data
# 4.1) calculate residual
# probability and store as matrix
response <- matrix(</pre>
  data = cbind(response, 1-response),
  nrow = length(actual)
)
# 4.2) calculate class-wise
# area under the curve
pr.auc(
  actual = actual,
  response = response
# 4.3) calculate class-wise
# weighted area under the curve
weighted.pr.auc(
  actual = actual,
  response = response,
```

```
= iris$Petal.Length/mean(iris$Petal.Length)
)
# 5) evaluate overall area under
# the curve
cat(
  "Micro-averaged area under the precision-recall curve", pr.auc(
             = actual,
   response = response,
   micro
             = TRUE
  "Micro-averaged area under the precision-recall curve (weighted)", weighted.pr.auc(
   actual
             = actual,
   response = response,
             = iris$Petal.Length/mean(iris$Petal.Length),
             = TRUE
   micro
 ),
 sep = "\n"
)
```

precision.factor

Precision

Description

A generic funcion for the precision. Use weighted.fdr() for the weighted precision.

Other names:

Positive Predictive Value

```
## S3 method for class 'factor'
precision(actual, predicted, micro = NULL, na.rm = TRUE, ...)

## S3 method for class 'factor'
weighted.precision(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)

## S3 method for class 'cmatrix'
precision(x, micro = NULL, na.rm = TRUE, ...)

## S3 method for class 'factor'
ppv(actual, predicted, micro = NULL, na.rm = TRUE, ...)

## S3 method for class 'factor'
weighted.ppv(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)

## S3 method for class 'cmatrix'
ppv(x, micro = NULL, na.rm = TRUE, ...)
```

```
## Generic S3 method
precision(
 . . . ,
micro = NULL,
na.rm = TRUE
)
## Generic S3 method
weighted.precision(
 . . . ,
W,
micro = NULL,
na.rm = TRUE
)
## Generic S3 method
ppv(
...,
micro = NULL,
na.rm = TRUE
## Generic S3 method
weighted.ppv(
 . . . ,
W,
micro = NULL,
na.rm = TRUE
)
```

Arguments

Χ

actual A vector of <factor> values of length n, and k levels. A vector of <factor> values of length n, and k levels. predicted A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro micro average across all k classes, if FALSE it returns the macro average. na.rm A <logical> value of length 1 (default: TRUE). If TRUE, NA values are removed from the computation. This argument is only relevant when micro!= NULL. When na.rm = TRUE, the computation corresponds to sum(c(1, 2, NA),na.rm = TRUE) / length(na.omit(c(1, 2, NA))). When na.rm = FALSE, the computation corresponds to sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA), na.rm = TRUE) / leng2, NA)). Arguments passed into other methods A <numeric>-vector of length n. NULL by default. W

A confusion matrix created cmatrix().

Value

If micro is NULL (the default), a named <numeric>-vector of length k
If micro is TRUE or FALSE, a <numeric>-vector of length 1

Definition

Let $\hat{\pi} \in [0,1]$ be the proportion of true positives among the predicted positives. The precision of the classifier is calculated as,

$$\hat{\pi} = \frac{\#TP_k}{\#TP_k + \#FP_k}$$

Where:

- $\#TP_k$ is the number of true positives, and
- $\#FP_k$ is the number of false positives.

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), recall.factor(), rase.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rrse.numeric(), specificity.factor(), zerooneloss.factor()
```

Examples

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
)
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
        = iris,
  family = binomial(
    link = "logit"
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
    predict(model, type = "response") > 0.5
  ),
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
```

90 preorder

```
# 4) evaluate class-wise performance
# using Precision
# 4.1) unweighted Precision
precision(
  actual
           = actual,
  predicted = predicted
# 4.2) weighted Precision
weighted.precision(
         = actual,
  actual
  predicted = predicted,
           = iris$Petal.Length/mean(iris$Petal.Length)
)
# 5) evaluate overall performance
# using micro-averaged Precision
cat(
  "Micro-averaged Precision", precision(
   actual
            = actual,
   predicted = predicted,
             = TRUE
   micro
  "Micro-averaged Precision (weighted)", weighted.precision(
   actual
            = actual,
   predicted = predicted,
             = iris$Petal.Length/mean(iris$Petal.Length),
   micro
             = TRUE
  ),
  sep = "\n"
)
```

preorder

Preorder

Description

This function does a column-wise ordering permutation of numeric or integer matrix.

```
preorder(
   x,
   decreasing = FALSE,
   ...
)
```

presort 91

Arguments

a numeric or integer matrix to be sorted.
 decreasing a logical value of length 1 (default: FALSE). If TRUE the matrix is returned in descending order.
 Arguments passed into other methods.

Value

A matrix with indices to the ordered values.

See Also

```
Other Tools: auc.numeric(), cov.wt.matrix(), presort()
```

Examples

```
# 1) generate a 4x4 matrix
# with random values to be sorted
set.seed(1903)
X <- matrix(
   data = cbind(sample(16:1)),
   nrow = 4
)
# 2) sort matrix
# in decreasing order
presort(X)
# 3) get indices
# for sorted matrix
preorder(X)</pre>
```

presort

Presort

Description

This generic function does a column-wise sorting of a numeric or integer matrix.

```
presort(
    x,
    decreasing = FALSE,
    ...
)
```

Arguments

```
    a numeric or integer matrix to be sorted.
    decreasing a logical value of length 1 (default: FALSE). If TRUE the matrix is returned in descending order.
    Arguments passed into other methods.
```

Value

A matrix with sorted rows.

See Also

```
Other Tools: auc.numeric(), cov.wt.matrix(), preorder()
```

Examples

```
# 1) generate a 4x4 matrix
# with random values to be sorted
set.seed(1903)
X <- matrix(
   data = cbind(sample(16:1)),
   nrow = 4
)
# 2) sort matrix
# in decreasing order
presort(X)
# 3) get indices
# for sorted matrix
preorder(X)</pre>
```

prROC.factor

Precision-Recall Curve

Description

The prROC()-function computes the precision() and recall() at thresholds provided by the response- or thresholds-vector. The function constructs a data.frame() grouped by k-classes where each class is treated as a binary classification problem.

```
## S3 method for class 'factor'
prROC(actual, response, thresholds = NULL, presorted = FALSE, ...)
## S3 method for class 'factor'
weighted.prROC(actual, response, w, thresholds = NULL, presorted = FALSE, ...)
```

```
## Generic S3 method
prROC(
    actual,
    response,
    thresholds = NULL,
    presorted = FALSE,
    ...
)

## Generic S3 method
weighted.prROC(
    actual,
    response,
    w,
    thresholds = NULL,
    presorted = FALSE,
    ...
)
```

Arguments

actual A vector of <factor> values of length n, and k levels.

response A $n \times k$ <numeric>-matrix. The estimated response probabilities for each class k.

thresholds An optional <numeric> vector of length n (default: NULL).

presorted A <logical>-value length 1 (default: FALSE). If TRUE the input will not be sorted by threshold.

... Arguments passed into other methods.

w A < numeric >- vector of length n. NULL by default.

Value

A data.frame on the following form,

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)
## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)
#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

Definition

Let $\hat{\sigma} \in [0, 1]$ be the proportion of true negatives among the actual negatives. The specificity of the classifier is calculated as,

$$\hat{\sigma} = \frac{\#TN_k}{\#TN_k + \#FP_k}$$

Where:

- $\#TN_k$ is the number of true negatives, and
- $\#FP_k$ is the number of false positives.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()
```

```
Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

Examples

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
 iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
 formula = species_num ~ Sepal.Length + Sepal.Width,
 data = iris,
 family = binomial(
   link = "logit"
 )
)
# 3) generate predicted
# classes
response <- predict(model, type = "response")</pre>
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
 levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 4) generate precision-recall
# data
# 4.1) calculate residual
# probability and store as matrix
response <- matrix(</pre>
 data = cbind(response, 1-response),
 nrow = length(actual)
# 4.2) generate precision-recall
# data
```

96 rae.numeric

```
roc <- prROC(</pre>
  actual = actual,
  response = response
)
# 5) plot by species
plot(roc)
# 5.1) summarise
summary(roc)
# 6) provide custom
# threholds
roc <- prROC(</pre>
  actual
             = actual,
  response = response,
  thresholds = seq(
    1,
    0,
    length.out = 20
  )
)
# 5) plot by species
plot(roc)
```

rae.numeric

Relative Absolute Error

Description

The rae()-function calculates the normalized relative absolute error between the predicted and observed <numeric> vectors. The weighted.rae() function computes the weighted relative absolute error.

```
## S3 method for class 'numeric'
rae(actual, predicted, ...)

## S3 method for class 'numeric'
weighted.rae(actual, predicted, w, ...)

## Generic S3 method
rae(
    actual,
    predicted,
    ...
)
```

rae.numeric 97

```
## Generic S3 method
weighted.rae(
  actual,
  predicted,
  w,
  ...
)
```

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. Arguments passed into other methods.

A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A <numeric> vector of length 1.

Definition

The Relative Absolute Error (RAE) is calculated as:

$$RAE = \frac{\sum_{i=1}^{n} |y_i - v_i|}{\sum_{i=1}^{n} |y_i - \bar{y}|}$$

Where y_i are the actual values, v_i are the predicted values, and \bar{y} is the mean of the actual values.

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), mse.numeric(), pinball.numeric(), rmse.numeric(), rmsle.numeric(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), pv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

Examples

```
# 1) fit a linear
# regression
model <- lm(
 mpg ~ .,
  data = mtcars
# 1.1) define actual
# and predicted values
# to measure performance
         <- mtcars$mpg
actual
predicted <- fitted(model)</pre>
# 2) evaluate in-sample model
# performance using Relative Absolute Error (RAE)
  "Relative Absolute Error", rae(
   actual
             = actual,
   predicted = predicted,
  "Relative Absolute Error (weighted)", weighted.rae(
   actual
             = actual,
   predicted = predicted,
              = mtcars$mpg/mean(mtcars$mpg)
  ),
  sep = "\n"
)
```

recall.factor

Recall

Description

A generic funcion for the Recall. Use weighted. fdr() for the weighted Recall.

Other names:

Sensitivity, True Positive Rate

```
## S3 method for class 'factor'
recall(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.recall(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
recall(x, micro = NULL, na.rm = TRUE, ...)
```

```
## S3 method for class 'factor'
sensitivity(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.sensitivity(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
sensitivity(x, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
tpr(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.tpr(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
tpr(x, micro = NULL, na.rm = TRUE, ...)
## Generic S3 method
recall(
micro = NULL,
na.rm = TRUE
## Generic S3 method
sensitivity(
micro = NULL,
na.rm = TRUE
## Generic S3 method
tpr(
micro = NULL,
na.rm = TRUE
## Generic S3 method
weighted.recall(
 . . . ,
W,
micro = NULL,
na.rm = TRUE
)
```

Arguments

actual	A vector of \prec factor \gt values of length n , and k levels.
predicted	A vector of \prec factor \gt values of length n , and k levels.
micro	A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro average across all k classes, if FALSE it returns the macro average.</logical>
na.rm	A <logical> value of length 1 (default: TRUE). If TRUE, NA values are removed from the computation. This argument is only relevant when micro!= NULL. When na.rm = TRUE, the computation corresponds to sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA))). When na.rm = FALSE, the computation corresponds to sum(c(1, 2, NA)), na.rm = TRUE) / length(c(1, 2, NA)).</logical>
	Arguments passed into other methods
W	A <numeric>-vector of length n. NULL by default.</numeric>
Х	A confusion matrix created cmatrix().

Value

```
If micro is NULL (the default), a named <numeric>-vector of length k
If micro is TRUE or FALSE, a <numeric>-vector of length 1
```

Definition

Let $\hat{\rho} \in [0,1]$ be the proportion of true positives among the actual positives. The recall of the classifier is calculated as,

$$\hat{\rho} = \frac{\#TP_k}{\#TP_k + \#FN_k}$$

Where:

- $\#TP_k$ is the number of true positives, and
- $\#FN_k$ is the number of false negatives.

Creating < factor >

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), roc.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

Examples

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
 family = binomial(
   link = "logit"
  )
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
 as.numeric(
   predict(model, type = "response") > 0.5
 levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
 labels = c("Virginica", "Others")
)
# 4) evaluate class-wise performance
# using Recall
# 4.1) unweighted Recall
recall(
  actual
          = actual,
  predicted = predicted
# 4.2) weighted Recall
weighted.recall(
  actual = actual,
 predicted = predicted,
            = iris$Petal.Length/mean(iris$Petal.Length)
)
```

rmse.numeric 103

```
# 5) evaluate overall performance
# using micro-averaged Recall
cat(
  "Micro-averaged Recall", recall(
   actual
             = actual,
   predicted = predicted,
             = TRUE
   micro
  ),
  "Micro-averaged Recall (weighted)", weighted.recall(
             = actual,
   actual
   predicted = predicted,
             = iris$Petal.Length/mean(iris$Petal.Length),
             = TRUE
   micro
  ),
  sep = "\n"
```

rmse.numeric

Root Mean Squared Error

Description

The rmse()-function computes the root mean squared error between the observed and predicted <numeric> vectors. The weighted.rmse() function computes the weighted root mean squared error.

```
## S3 method for class 'numeric'
rmse(actual, predicted, ...)

## S3 method for class 'numeric'
weighted.rmse(actual, predicted, w, ...)

## Generic S3 method
rmse(
    actual,
    predicted,
    ...
)

weighted.rmse(
    actual,
    predicted,
    ...
)
```

104 rmse.numeric

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. Arguments passed into other methods.

W A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A <numeric> vector of length 1.

Definition

The metric is calculated as,

$$\sqrt{\frac{1}{n}\sum_{i}^{n}(y_{i}-v_{i})^{2}}$$

Where y_i and v_i are the actual and predicted values respectively.

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), mse.numeric(), pinball.numeric(), rae.numeric(), rmsle.numeric(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

Examples

```
# 1) fit a linear
# regression
model <- lm(
   mpg ~ .,
   data = mtcars
)
# 1.1) define actual
# and predicted values
# to measure performance
actual <- mtcars$mpg
predicted <- fitted(model)</pre>
```

rmsle.numeric 105

```
# 2) evaluate in-sample model
# performance using Root Mean Squared Error (RMSE)
cat(
   "Root Mean Squared Error", rmse(
    actual = actual,
    predicted = predicted,
),
   "Root Mean Squared Error (weighted)", weighted.rmse(
    actual = actual,
    predicted = predicted,
    w = mtcars$mpg/mean(mtcars$mpg)
),
   sep = "\n"
)
```

rmsle.numeric

Root Mean Squared Logarithmic Error

Description

The rmsle()-function computes the root mean squared logarithmic error between the observed and predicted <numeric> vectors. The weighted.rmsle() function computes the weighted root mean squared logarithmic error.

```
## S3 method for class 'numeric'
rmsle(actual, predicted, ...)

## S3 method for class 'numeric'
weighted.rmsle(actual, predicted, w, ...)

## Generic S3 method
rmsle(
    actual,
    predicted,
    ...
)

## Generic S3 method
weighted.rmsle(
    actual,
    predicted,
    w,
    ...
)
```

106 rmsle.numeric

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. Arguments passed into other methods.

W A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A <numeric> vector of length 1.

Definition

The metric is calculated as,

$$\sqrt{\frac{1}{n} \sum_{i=0}^{n} (\log(1+y_i) - \log(1+v_i))^2}$$

Where y_i and v_i are the actual and predicted values respectively.

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), mse.numeric(), pinball.numeric(), rae.numeric(), rmse.numeric(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

Examples

```
# 1) fit a linear
# regression
model <- lm(
   mpg ~ .,
   data = mtcars
)
# 1.1) define actual
# and predicted values
# to measure performance
actual <- mtcars$mpg
predicted <- fitted(model)</pre>
```

roc.auc.matrix 107

```
# 2) evaluate in-sample model
# performance using Root Mean Squared Logarithmic Error (RMSLE)
cat(
    "Root Mean Squared Logarithmic Error", rmsle(
        actual = actual,
        predicted = predicted,
    ),
    "Root Mean Squared Logarithmic Error (weighted)", weighted.rmsle(
        actual = actual,
        predicted = predicted,
        w = mtcars$mpg/mean(mtcars$mpg)
    ),
    sep = "\n"
)
```

roc.auc.matrix

Area under the Receiver Operator Characteristics Curve

Description

A generic function for the area under the Receiver Operator Characteristics Curve. Use weighted.roc.auc() for the weighted area under the Receiver Operator Characteristics Curve.

```
## S3 method for class 'matrix'
roc.auc(actual, response, micro = NULL, method = 0L, ...)
## S3 method for class 'matrix'
weighted.roc.auc(actual, response, w, micro = NULL, method = 0L, ...)
## Generic S3 method
roc.auc(
 actual,
response,
micro = NULL,
method = 0,
## Generic S3 method
weighted.roc.auc(
 actual,
 response,
micro = NULL,
 method = 0,
```

108 roc.auc.matrix

)

Arguments

actual A vector of <factor> values of length n, and k levels.

response A $n \times k$ < numeric > matrix. The estimated response probabilities for each class

k.

micro A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro

average across all k classes, if FALSE it returns the macro average.

method A <numeric> value (default: 0). Defines the underlying method of calculating

the area under the curve. If 0 it is calculated using the trapezoid-method, if 1

it is calculated using the step-method.

... Arguments passed into other methods.

w A < numeric >- vector of length n. NULL by default.

Value

A <numeric> vector of length 1

Definition

Trapezoidal rule

The **trapezoidal rule** approximates the integral of a function f(x) between x = a and x = b using trapezoids formed between consecutive points. If we have points x_0, x_1, \ldots, x_n (with $a = x_0 < x_1 < \cdots < x_n = b$) and corresponding function values $f(x_0), f(x_1), \ldots, f(x_n)$, the area under the curve A_T is approximated by:

$$A_T \approx \sum_{k=1}^{n} \frac{f(x_{k-1}) + f(x_k)}{2} [x_k - x_{k-1}].$$

Step-function method

The **step-function** (**rectangular**) **method** uses the value of the function at one endpoint of each subinterval to form rectangles. With the same partition x_0, x_1, \ldots, x_n , the rectangular approximation A_S can be written as:

$$A_S \approx \sum_{k=1}^{n} f(x_{k-1}) [x_k - x_{k-1}].$$

See Also

Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), specificity.factor(), zerooneloss.factor()

roc.auc.matrix 109

```
Other Supervised Learning: ROC. factor(), accuracy. factor(), baccuracy. factor(), ccc.numeric(),
ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(),
fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(),
mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(),
npv.factor(),pinball.numeric(),plr.factor(),pr.auc.matrix(),prROC.factor(),precision.factor(),
rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), rrmse.numeric(), rrse.numeric(),
rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(),
cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(),
fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(),
npv.factor(),plr.factor(),pr.auc.matrix(),prROC.factor(),precision.factor(),recall.factor(),
specificity.factor(), zerooneloss.factor()
Other Supervised Learning: ROC. factor(), accuracy. factor(), baccuracy. factor(), ccc.numeric(),
ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(),
fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(),
mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(),
npv.factor(),pinball.numeric(),plr.factor(),pr.auc.matrix(),prROC.factor(),precision.factor(),
rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), rrmse.numeric(), rrse.numeric(),
rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
)
# 2) fit the logistic
# regression
model <- glm(
  formula = species_num ~ Sepal.Length + Sepal.Width,
        = iris,
  family = binomial(
    link = "logit"
  )
)
# 3) generate predicted
# classes
response <- predict(model, type = "response")</pre>
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
```

```
# 4) generate receiver operator characteristics
# 4.1) calculate residual
# probability and store as matrix
response <- matrix(</pre>
 data = cbind(response, 1-response),
 nrow = length(actual)
)
# 4.2) calculate class-wise
# area under the curve
roc.auc(
 actual
          = actual,
 response = response
# 4.3) calculate class-wise
# weighted area under the curve
weighted.roc.auc(
 actual = actual,
 response = response,
          = iris$Petal.Length/mean(iris$Petal.Length)
)
# 5) evaluate overall area under
# the curve
cat(
  "Micro-averaged area under the ROC curve", roc.auc(
   actual = actual,
   response = response,
   micro
             = TRUE
  "Micro-averaged area under the ROC curve (weighted)", weighted.roc.auc(
             = actual,
   response = response,
             = iris$Petal.Length/mean(iris$Petal.Length),
             = TRUE
   micro
 ),
 sep = "\n"
```

ROC.factor

Receiver Operator Characteristics

Description

The ROC()-function computes the tpr() and fpr() at thresholds provided by the response- or thresholds-vector. The function constructs a data.frame() grouped by k-classes where each class is treated as a binary classification problem.

Usage

```
## S3 method for class 'factor'
ROC(actual, response, thresholds = NULL, presorted = FALSE, ...)
## S3 method for class 'factor'
weighted.ROC(actual, response, w, thresholds = NULL, presorted = FALSE, ...)
## Generic S3 method
ROC(
 actual,
 response,
 thresholds = NULL,
 presorted = FALSE,
## Generic S3 method
weighted.ROC(
 actual,
 response,
 W,
 thresholds = NULL,
 presorted = FALSE,
)
```

Arguments

actual A vector of <factor> values of length n, and k levels.

Response A $n \times k$ <numeric>-matrix. The estimated response probabilities for each class k.

An optional <numeric> vector of length n (default: NULL).

Presorted A <logical>-value length 1 (default: FALSE). If TRUE the input will not be sorted by threshold.

Arguments passed into other methods.

A <numeric>-vector of length n. NULL by default.

Value

A data.frame on the following form,

```
threshold <numeric> Thresholds used to determine tpr() and fpr()
level <character> The level of the actual <factor>
label <character> The levels of the actual <factor>
fpr <numeric> The false positive rate
tpr <numeric> The true positive rate
```

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

Definition

Let $\hat{\sigma} \in [0, 1]$ be the proportion of true negatives among the actual negatives. The specificity of the classifier is calculated as,

$$\hat{\sigma} = \frac{\#TN_k}{\#TN_k + \#FP_k}$$

Where:

- $\#TN_k$ is the number of true negatives, and
- $\#FP_k$ is the number of false positives.

See Also

```
Other Classification: accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor(), zerooneloss.factor()

Other Supervised Learning: accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
 iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
 formula = species_num ~ Sepal.Length + Sepal.Width,
        = iris,
 family = binomial(
   link = "logit"
# 3) generate predicted
# classes
response <- predict(model, type = "response")</pre>
# 3.1) generate actual
# classes
actual <- factor(</pre>
 x = iris$species_num,
 levels = c(1,0),
 labels = c("Virginica", "Others")
# 4) generate reciever
# operator characteristics
# 4.1) calculate residual
# probability and store as matrix
```

114 rrmse.numeric

```
response <- matrix(</pre>
 data = cbind(response, 1-response),
 nrow = length(actual)
# 4.2) construct
# data.frame
roc <- ROC(
 actual = actual,
 response = response
# 5) plot by species
plot(roc)
# 5.1) summarise
summary(roc)
# 6) provide custom
# threholds
roc <- ROC(
 actual
             = actual,
 response = response,
 thresholds = seq(
   1,
   0,
   length.out = 20
 )
)
# 5) plot by species
plot(roc)
```

rrmse.numeric

Relative Root Mean Squared Error

Description

The rrmse()-function computes the Relative Root Mean Squared Error between the observed and predicted <numeric> vectors. The weighted.rrmse() function computes the weighted Relative Root Mean Squared Error.

Usage

```
## S3 method for class 'numeric'
rrmse(actual, predicted, normalization = 1L, ...)
## S3 method for class 'numeric'
weighted.rrmse(actual, predicted, w, normalization = 1L, ...)
```

rrmse.numeric 115

```
## Generic S3 method
rrmse(
  actual,
  predicted,
  normalization = 1,
  ...
)

## Generic S3 method
weighted.rrmse(
  actual,
  predicted,
  w,
  normalization = 1,
  ...
)
```

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. Normalization A <numeric>-value of length 1 (default: 1). 0: mean-normalization, 1: range-normalization, 2: IQR-normalization.

Arguments passed into other methods.

A <numeric>-vector of length n. The weight assigned to each observation in the data

Value

A <numeric> vector of length 1.

Definition

The metric is calculated as,

$$\frac{RMSE}{\gamma}$$

Where γ is the normalization factor.

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), mse.numeric(), pinball.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(),
```

116 rrse.numeric

```
fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(),
mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(),
npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(),
rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(),
rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

Examples

```
# 1) fit a linear
# regression
model <- lm(
 mpg ~ .,
 data = mtcars
# 1.1) define actual
# and predicted values
# to measure performance
          <- mtcars$mpg
predicted <- fitted(model)</pre>
# 2) evaluate in-sample model
# performance using Relative Root Mean Squared Error (RRMSE)
cat(
  "IQR Relative Root Mean Squared Error", rrmse(
                 = actual,
   actual
   predicted
                  = predicted,
   normalization = 2
  "IQR Relative Root Mean Squared Error (weighted)", weighted.rrmse(
   actual
                 = actual,
   predicted
                  = predicted,
                  = mtcars$mpg/mean(mtcars$mpg),
   normalization = 2
 ),
 sep = "\n"
)
```

rrse.numeric

Root Relative Squared Error

Description

The rrse()-function calculates the root relative squared error between the predicted and observed <numeric> vectors. The weighted.rrse() function computes the weighted root relative squared errorr.

rrse.numeric 117

Usage

```
## S3 method for class 'numeric'
rrse(actual, predicted, ...)

## S3 method for class 'numeric'
weighted.rrse(actual, predicted, w, ...)

## Generic S3 method
rrse(
    actual,
    predicted,
    ...
)

## Generic S3 method
weighted.rrse(
    actual,
    predicted,
    w,
    ...
)
```

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. predicted A <numeric>-vector of length n. The estimated (continuous) response variable. ... Arguments passed into other methods. W A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A <numeric> vector of length 1.

Definition

The metric is calculated as,

RRSE =
$$\sqrt{\frac{\sum_{i=1}^{n}(y_i - v_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Where y_i are the actual values, v_i are the predicted values, and \bar{y} is the mean of the actual values.

118 rsq.numeric

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), mse.numeric(), pinball.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rmsle.numeric(), rmsle.numeric(), rmsle.numeric(), rmsle.numeric(), ccc.numeric(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

Examples

```
# 1) fit a linear
# regression
model <- lm(
  mpg ~ .,
  data = mtcars
# 1.1) define actual
# and predicted values
# to measure performance
actual <- mtcars$mpg</pre>
predicted <- fitted(model)</pre>
# 2) evaluate in-sample model
# performance using Relative Root Squared Errror (RRSE)
cat(
  "Relative Root Squared Errror", rrse(
   actual = actual,
   predicted = predicted,
  "Relative Root Squared Errror (weighted)", weighted.rrse(
   actual
            = actual,
   predicted = predicted,
              = mtcars$mpg/mean(mtcars$mpg)
  ),
  sep = "\n"
)
```

rsq.numeric

 R^2

Description

A generic function for the \mathbb{R}^2 . The unadjusted \mathbb{R}^2 is returned by default. Use weighted rsq() for the weighted \mathbb{R}^2 .

rsq.numeric 119

Usage

Arguments

Value

A <numeric> vector of length 1.

Definition

Let $R^2 \in [-\infty, 1]$ be the explained variation. The R^2 is calculated as,

$$R^{2} = 1 - \frac{\sum (y_{i} - \hat{y}_{i})^{2}}{\sum (y_{i} - \bar{y})^{2}} \frac{n - 1}{n - (k + 1)}$$

Where:

- n is the number of observations
- *k* is the number of features
- y is the actual values
- \hat{y}_i is the predicted values

120 rsq.numeric

- $\sum (y_i \hat{y}_i)^2$ is the sum of squared errors and,
- $\sum (y_i \bar{y})^2$ is total sum of squared errors

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), mse.numeric(), pinball.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rrmse.numeric(), rrmse.numeric(), smape.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), smape.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) fit a linear
# regression
model <- lm(
  mpg ~ .,
  data = mtcars
)
# 1.1) define actual
# and predicted values
# to measure in-sample performance
actual
         <- mtcars$mpg
predicted <- fitted(model)</pre>
# 2) calculate performance
# using R squared adjusted and
# unadjused for features
cat(
  "Rsq", rsq(
   actual
             = actual,
   predicted = fitted(model)
  "Rsq (Adjusted)", rsq(
   actual = actual,
   predicted = fitted(model),
   k = ncol(model.matrix(model)) - 1
  ),
  sep = "\n"
```

smape.numeric 121

smape.numeric

Symmetric Mean Absolutte Percentage Error

Description

The smape()-function computes the symmetric mean absolute percentage error between the observed and predicted <numeric> vectors. The weighted.smape() function computes the weighted symmetric mean absolute percentage error.

Usage

```
## S3 method for class 'numeric'
smape(actual, predicted, ...)

## S3 method for class 'numeric'
weighted.smape(actual, predicted, w, ...)

## Generic S3 method
smape(
    actual,
    predicted,
    ...
)

## Generic S3 method
weighted.smape(
    actual,
    predicted,
    w,
    ...
)
```

Arguments

actual A <numeric>-vector of length n. The observed (continuous) response variable. A <numeric>-vector of length n. The estimated (continuous) response variable. Arguments passed into other methods. A <numeric>-vector of length n. The weight assigned to each observation in the data.

Value

A < numeric > vector of length 1.

122 smape.numeric

Definition

The metric is calculated as follows,

$$\sum_{i}^{n} \frac{1}{n} \frac{|y_i - v_i|}{\frac{|y_i| + |v_i|}{2}}$$

where y_i and v_i is the actual and predicted values respectively.

See Also

```
Other Regression: ccc.numeric(), huberloss.numeric(), mae.numeric(), mape.numeric(), mpe.numeric(), mse.numeric(), pinball.numeric(), rae.numeric(), rmse.numeric(), rmsle.numeric(), rrmse.numeric(), rrse.numeric(), rsq.numeric()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(), rrmse.numeric(), rrse.numeric(), rsq.numeric(), specificity.factor(), zerooneloss.factor()
```

```
# 1) fit a linear
# regression
model <- lm(
 mpg ~ .,
 data = mtcars
# 1.1) define actual
# and predicted values
# to measure performance
         <- mtcars$mpg
predicted <- fitted(model)</pre>
# 2) evaluate in-sample model
# performance using Symmetric Mean Absolute Percentage Error (MAPE)
cat(
  "Symmetric Mean Absolute Percentage Error", mape(
   actual = actual,
   predicted = predicted,
 ),
  "Symmetric Mean Absolute Percentage Error (weighted)", weighted.mape(
   actual = actual,
   predicted = predicted,
             = mtcars$mpg/mean(mtcars$mpg)
 ),
 sep = "\n"
```

```
specificity.factor Specificity
```

Description

A generic funcion for the Specificity. Use weighted specificity() for the weighted Specificity.

Other names:

True Negative Rate, Selectivity

Usage

```
## S3 method for class 'factor'
specificity(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.specificity(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
specificity(x, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
tnr(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.tnr(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
tnr(x, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
selectivity(actual, predicted, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'factor'
weighted.selectivity(actual, predicted, w, micro = NULL, na.rm = TRUE, ...)
## S3 method for class 'cmatrix'
selectivity(x, micro = NULL, na.rm = TRUE, ...)
## Generic S3 method
specificity(
. . . ,
micro = NULL,
na.rm = TRUE
)
## Generic S3 method
```

```
tnr(
micro = NULL,
na.rm = TRUE
## Generic S3 method
selectivity(
...,
micro = NULL,
na.rm = TRUE
)
## Generic S3 method
weighted.specificity(
. . . ,
W,
micro = NULL,
na.rm = TRUE
## Generic S3 method
weighted.tnr(
. . . ,
W,
micro = NULL,
na.rm = TRUE
)
## Generic S3 method
weighted.selectivity(
. . . ,
W,
micro = NULL,
na.rm = TRUE
)
```

Arguments

actual	A vector of \prec factor \gt values of length n , and k levels.
predicted	A vector of $<$ factor $>$ values of length n , and k levels.
micro	A <logical>-value of length 1 (default: NULL). If TRUE it returns the micro average across all k classes, if FALSE it returns the macro average.</logical>
na.rm	A <logical> value of length 1 (default: TRUE). If TRUE, NA values are removed from the computation. This argument is only relevant when micro != NULL. When na.rm = TRUE, the computation corresponds to $sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))$. When na.rm = FALSE, the</logical>

```
computation corresponds to sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA)).

... Arguments passed into other methods

w A <numeric>-vector of length n. NULL by default.

x A confusion matrix created cmatrix().
```

Value

```
If micro is NULL (the default), a named <numeric>-vector of length k

If micro is TRUE or FALSE, a <numeric>-vector of length 1
```

Creating < factor >

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

Definition

Let $\hat{\sigma} \in [0, 1]$ be the proportion of true negatives among the actual negatives. The specificity of the classifier is calculated as,

$$\hat{\sigma} = \frac{\#TN_k}{\#TN_k + \#FP_k}$$

Where:

- $\#TN_k$ is the number of true negatives, and
- $\#FP_k$ is the number of false positives.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), zerooneloss.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fpr.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(), roc.auc.matrix(), rrae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), zerooneloss.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(
    iris$Species == "virginica"
)

# 2) fit the logistic
# regression
model <- glm(
    formula = species_num ~ Sepal.Length + Sepal.Width,
    data = iris,
    family = binomial(
        link = "logit"
    )
)

# 3) generate predicted
# classes
predicted <- factor(
    as.numeric(</pre>
```

wine_quality 127

```
predict(model, type = "response") > 0.5
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 4) evaluate class-wise performance
# using Specificity
# 4.1) unweighted Specificity
specificity(
  actual
          = actual,
  predicted = predicted
)
# 4.2) weighted Specificity
weighted.specificity(
  actual = actual,
  predicted = predicted,
           = iris$Petal.Length/mean(iris$Petal.Length)
)
# 5) evaluate overall performance
# using micro-averaged Specificity
  "Micro-averaged Specificity", specificity(
   actual
             = actual,
   predicted = predicted,
             = TRUE
   micro
  ),
  "Micro-averaged Specificity (weighted)", weighted.specificity(
   actual
            = actual,
   predicted = predicted,
   W
             = iris$Petal.Length/mean(iris$Petal.Length),
   micro
              = TRUE
  ),
  sep = "\n"
)
```

128 wine_quality

Description

This dataset contains measurements of various chemical properties of white wines along with their quality ratings and a quality classification. The dataset was obtained from the UCI Machine Learning Repository.

Usage

```
data(wine_quality)
```

Format

A list with two components:

features A data frame with 11 chemical property variables.

target A list with two elements: regression (wine quality scores) and class (quality classification).

Details

The data is provided as a list with two components:

```
features A data frame containing the chemical properties of the wines. The variables include:
```

```
fixed_acidity Fixed acidity (g/L).

volatile_acidity Volatile acidity (g/L), mainly due to acetic acid.

citric_acid Citric acid (g/L).

residual_sugar Residual sugar (g/L).

chlorides Chloride concentration (g/L).

free_sulfur_dioxide Free sulfur dioxide (mg/L).

total_sulfur_dioxide Total sulfur dioxide (mg/L).

density Density of the wine (g/cm³).

pH pH value of the wine.

sulphates Sulphates (g/L).

alcohol Alcohol content (% by volume).
```

target A list containing two elements:

regression A numeric vector representing the wine quality scores (used as the regression target).

class A factor with levels "High Quality", "Medium Quality", and "Low Quality", where classification is determined as follows:

```
High Quality quality \geq 7.

Low Quality quality \leq 4.

Medium Quality for all other quality scores.
```

Source

```
https://archive.ics.uci.edu/dataset/186/wine+quality
```

zerooneloss.factor Zero-One Loss

Description

The zerooneloss()-function computes the zero-one Loss, a classification loss function that calculates the proportion of misclassified instances between two vectors of predicted and observed factor() values. The weighted.zerooneloss() function computes the weighted zero-one loss.

Usage

```
## S3 method for class 'factor'
zerooneloss(actual, predicted, ...)
## S3 method for class 'factor'
weighted.zerooneloss(actual, predicted, w, ...)
## S3 method for class 'cmatrix'
zerooneloss(x, ...)
## Generic S3 method
zerooneloss(...)
## Generic S3 method
weighted.zerooneloss(
...,
w
)
```

Arguments

actual A vector of <factor> with length n, and k levels predicted A vector of <factor> with length n, and k levels ... Arguments passed into other methods w A <numeric>-vector of length n. NULL by default x A confusion matrix created cmatrix()

Value

A <numeric>-vector of length 1

Definition

The metric is calculated as follows,

$$\frac{\#FP+\#FN}{\#TP+\#TN+\#FP+\#FN}$$

Where #TP, #TN, #FP, and #FN represent the true positives, true negatives, false positives, and false negatives, respectively.

Creating <factor>

Consider a classification problem with three classes: A, B, and C. The actual vector of factor() values is defined as follows:

```
## set seed
set.seed(1903)

## actual
factor(
    x = sample(x = 1:3, size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] B A B B A C B C C A
#> Levels: A B C
```

Here, the values 1, 2, and 3 are mapped to A, B, and C, respectively. Now, suppose your model does not predict any B's. The predicted vector of factor() values would be defined as follows:

```
## set seed
set.seed(1903)

## predicted
factor(
    x = sample(x = c(1, 3), size = 10, replace = TRUE),
    levels = c(1, 2, 3),
    labels = c("A", "B", "C")
)

#> [1] C A C C C C C C A C
#> Levels: A B C
```

In both cases, k = 3, determined indirectly by the levels argument.

See Also

```
Other Classification: ROC.factor(), accuracy.factor(), baccuracy.factor(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), fmi.factor(), fpr.factor(), jaccard.factor(), logloss.factor(), mcc.factor(), nlr.factor(), npv.factor(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(), recall.factor(), roc.auc.matrix(), specificity.factor()

Other Supervised Learning: ROC.factor(), accuracy.factor(), baccuracy.factor(), ccc.numeric(), ckappa.factor(), cmatrix.factor(), dor.factor(), entropy.matrix(), fbeta.factor(), fdr.factor(), fer.factor(), huberloss.numeric(), jaccard.factor(), logloss.factor(), mae.numeric(), mape.numeric(), mcc.factor(), mpe.numeric(), mse.numeric(), nlr.factor(),
```

```
npv.factor(), pinball.numeric(), plr.factor(), pr.auc.matrix(), prROC.factor(), precision.factor(),
rae.numeric(), recall.factor(), rmse.numeric(), rmsle.numeric(), roc.auc.matrix(),
rrmse.numeric(), rrse.numeric(), rsq.numeric(), smape.numeric(), specificity.factor()
```

```
# 1) recode Iris
# to binary classification
# problem
iris$species_num <- as.numeric(</pre>
  iris$Species == "virginica"
# 2) fit the logistic
# regression
model <- glm(</pre>
  formula = species_num ~ Sepal.Length + Sepal.Width,
  data = iris,
  family = binomial(
   link = "logit"
)
# 3) generate predicted
# classes
predicted <- factor(</pre>
  as.numeric(
   predict(model, type = "response") > 0.5
  ),
 levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 3.1) generate actual
# classes
actual <- factor(</pre>
  x = iris$species_num,
  levels = c(1,0),
  labels = c("Virginica", "Others")
)
# 4) evaluate model
# performance using Zero-One Loss
cat(
  "Zero-One Loss", zerooneloss(
   actual = actual,
   predicted = predicted
  "Zero-One Loss (weighted)", weighted.zerooneloss(
            = actual,
   actual
   predicted = predicted,
            = iris$Petal.Length/mean(iris$Petal.Length)
  ),
```

```
sep = "\n"
)
```

Index

* Classification	rsq.numeric, 118
accuracy.factor, 3	smape.numeric, 121
baccuracy.factor,8	* Supervised Learning
ckappa.factor, 14	accuracy.factor, 3
cmatrix.factor, 18	baccuracy.factor, 8
dor.factor, 21	ccc.numeric, 12
entropy.matrix, 25	ckappa.factor, 14
fbeta.factor, 27	cmatrix.factor, 18
fdr.factor, 31	dor.factor,21
fer.factor, 35	entropy.matrix, 25
fmi.factor, 39	fbeta.factor, 27
fpr.factor, 41	fdr.factor, 31
jaccard.factor,48	fer.factor, 35
logloss.factor, 53	fpr.factor,41
mcc.factor, 60	huberloss.numeric, 46
nlr.factor,68	jaccard.factor,48
npv.factor, 71	logloss.factor, 53
plr.factor,79	mae.numeric, 56
pr.auc.matrix,82	mape.numeric, 58
precision.factor,86	mcc.factor, 60
prROC.factor,92	mpe.numeric,64
recall.factor,98	mse.numeric,66
roc.auc.matrix, 107	nlr.factor,68
ROC.factor, 110	npv.factor,71
specificity.factor, 123	pinball.numeric,77
zerooneloss.factor, 129	plr.factor,79
* Regression	pr.auc.matrix,82
ccc.numeric, 12	precision.factor, 86
huberloss.numeric, 46	prROC.factor,92
mae.numeric,56	rae.numeric,96
mape.numeric,58	recall.factor,98
mpe.numeric, 64	rmse.numeric, 103
mse.numeric, 66	rmsle.numeric, 105
pinball.numeric,77	roc.auc.matrix, 107
rae.numeric,96	ROC.factor, 110
rmse.numeric, 103	rrmse.numeric, 114
rmsle.numeric, 105	rrse.numeric, 116
rrmse.numeric, 114	rsq.numeric, 118
rrse.numeric, 116	smape.numeric, 121

specificity.factor, 123	63, 65, 67, 70, 73, 78, 81, 84, 89, 94,
zerooneloss.factor, 129	95, 97, 101, 104, 106, 108, 109, 113,
* Tools	115, 118, 120, 122, 126, 130
auc.numeric, 6	cov.wt.matrix, 7, 91, 92
preorder, 90	<pre>cross.entropy(entropy.matrix), 25</pre>
presort, 91	csi (jaccard.factor),48
* Unsupervised Learning	
fmi.factor, 39	data.frame, <i>93</i> , <i>111</i>
* datasets	data.frame(), 92, 110
banknote, 11	dor (dor.factor), 21
obesity, 75	dor.factor, 5, 10, 13, 16, 20, 21, 26, 30, 33,
<pre>wine_quality, 127</pre>	34, 37, 40, 44, 47, 51, 55, 58, 60, 63,
	65, 67, 70, 73, 78, 81, 84, 89, 94, 95,
accuracy (accuracy.factor), 3	97, 101, 104, 106, 108, 109, 113,
accuracy.factor, 3, 10, 13, 16, 20, 23, 26,	115, 118, 120, 122, 126, 130
30, 33, 34, 37, 40, 44, 47, 51, 55, 58,	
60, 63, 65, 67, 70, 73, 78, 81, 84, 89,	entropy(entropy.matrix), 25
94, 95, 97, 101, 104, 106, 108, 109,	entropy(), 25
113, 115, 118, 120, 122, 126, 130	entropy.matrix, 5, 10, 13, 16, 20, 23, 25, 30,
auc(auc.numeric),6	33, 34, 37, 40, 44, 47, 51, 55, 58, 60,
auc(), <u>6</u>	63, 65, 67, 70, 73, 78, 81, 84, 89, 94,
auc.numeric, 6, <i>91</i> , <i>92</i>	95, 97, 101, 104, 106, 108, 109, 113,
	115, 118, 120, 122, 126, 130
baccuracy (baccuracy $.$ factor), 8	
baccuracy.factor, 5, 8, 13, 16, 20, 23, 26,	factor, 3, 4, 8, 9, 15, 16, 18, 19, 22, 23, 28,
30, 33, 34, 37, 40, 44, 47, 51, 55, 58,	29, 32, 33, 36, 37, 39, 40, 43, 44, 50,
60, 63, 65, 67, 70, 73, 78, 81, 84, 89,	51, 53, 54, 61, 62, 68, 69, 72, 80, 83,
94, 95, 97, 101, 104, 106, 108, 109,	87, 88, 93, 100, 101, 108, 111, 112,
113, 115, 118, 120, 122, 126, 130	124, 125, 129, 130
banknote, 11	factor(), 4, 9, 10, 16, 19, 23, 29, 33, 37, 39,
.) 10	40, 44, 48, 51, 54, 55, 61, 62, 69,
ccc (ccc.numeric), 12	71–73, 80, 81, 88, 93, 94, 101, 112,
ccc.numeric, 5, 10, 12, 16, 20, 23, 26, 30, 34,	125, 129, 130
<i>37</i> , <i>44</i> , <i>47</i> , <i>51</i> , <i>55</i> , <i>58</i> , <i>60</i> , <i>63</i> , <i>65</i> , <i>67</i> ,	fallout (fpr.factor), 41
70, 73, 78, 81, 84, 89, 95, 97, 101,	FALSE, 6, 9, 13, 15, 28, 32, 36, 43, 50, 69, 72,
104, 106, 109, 113, 115, 118, 120,	78, 80, 83, 87, 88, 91–93, 100, 108,
122, 126, 130	111, 124, 125
character, 93, 111	fbeta (fbeta. factor), 27
ckappa (ckappa. factor), 14	fbeta.factor, 5, 10, 13, 16, 20, 23, 26, 27,
ckappa.factor, 5, 10, 13, 14, 20, 23, 26, 30,	33, 34, 37, 40, 44, 47, 51, 55, 58, 60,
33, 34, 37, 40, 44, 47, 51, 55, 58, 60,	63, 65, 67, 70, 73, 78, 81, 84, 89, 94,
63, 65, 67, 70, 73, 78, 81, 84, 89, 94,	95, 97, 101, 104, 106, 108, 109, 113,
95, 97, 101, 104, 106, 108, 109, 113,	115, 118, 120, 122, 126, 130
115, 118, 120, 122, 126, 130	fdr (fdr. factor), 31
cmatrix (cmatrix.factor), 18	fdr.factor, 5, 10, 13, 16, 20, 23, 26, 30, 31,
cmatrix(), 3, 9, 15, 18, 22, 28, 32, 36, 39, 43,	37, 40, 44, 47, 51, 55, 58, 60, 63, 65, 67, 70, 73, 78, 81, 84, 80, 04, 05, 07
50, 62, 69, 72, 80, 87, 100, 125, 129 cmatrix.factor, 5, 10, 13, 16, 18, 23, 26, 30,	67, 70, 73, 78, 81, 84, 89, 94, 95, 97,
	101, 104, 106, 108, 109, 113, 116,
33, 34, 37, 40, 44, 47, 51, 55, 58, 60,	118, 120, 122, 126, 130

fer (fer. factor), 35	63, 65, 67, 70, 73, 78, 81, 84, 89, 94,
fer.factor, 5, 10, 13, 16, 20, 23, 26, 30, 33,	95, 97, 101, 104, 106, 108, 109, 113,
34, 35, 40, 44, 47, 51, 55, 58, 60, 63,	116, 118, 120, 122, 126, 130
65, 67, 70, 73, 78, 81, 84, 89, 94, 95,	
97, 101, 104, 106, 108, 109, 113,	mae (mae.numeric), 56
116, 118, 120, 122, 126, 130	mae(), 57
fmi (fmi.factor), 39	mae.numeric, 5, 10, 13, 16, 20, 23, 26, 30, 34,
fmi(), 39	37, 44, 47, 51, 55, 56, 60, 63, 65, 67,
fmi.factor, 5, 10, 16, 20, 23, 26, 30, 33, 37,	70, 73, 78, 81, 84, 89, 95, 97, 101,
39, 44, 51, 55, 63, 70, 73, 81, 84, 89,	104, 106, 109, 113, 115, 116, 118,
94, 101, 108, 109, 113, 126, 130	120, 122, 126, 130
fpr (fpr.factor), 41	mape (mape.numeric), 58
fpr(), 110, 111	mape(), 59
fpr.factor, 5, 10, 13, 16, 20, 23, 26, 30, 33,	mape.numeric, 5, 10, 13, 17, 20, 23, 26, 30,
34, 37, 40, 41, 47, 51, 55, 58, 60, 63,	34, 37, 44, 47, 51, 55, 58, 58, 63, 65,
65, 67, 70, 73, 78, 81, 84, 89, 94, 95,	67, 70, 73, 78, 81, 84, 89, 95, 97,
97, 101, 104, 106, 108, 109, 113,	101, 104, 106, 109, 113, 115, 116,
116, 118, 120, 122, 126, 130	118, 120, 122, 126, 130
	matrix, 18, 19, 83, 91–93, 108, 111
huberloss (huberloss.numeric), 46	mcc (mcc.factor), 60
huberloss(), 46	mcc(), 61
huberloss.numeric, 5, 10, 13, 16, 20, 23, 26,	mcc.factor, 5, 10, 13, 16, 17, 20, 23, 26, 30,
30, 34, 37, 44, 46, 51, 55, 58, 60, 63,	33, 34, 37, 40, 44, 47, 51, 55, 58, 60,
65, 67, 70, 73, 78, 81, 84, 89, 95, 97,	60, 65, 67, 70, 73, 78, 81, 84, 89, 94,
101, 104, 106, 109, 113, 115, 116,	95, 97, 101, 104, 106, 108, 109, 113,
118, 120, 122, 126, 130	116, 118, 120, 122, 126, 130
integer, 26, 76, 90-92	mean, 115
IQR, <i>115</i>	mpe (mpe.numeric), 64
	mpe(), 64
jaccard(jaccard.factor),48	mpe.numeric, 5, 10, 13, 17, 20, 23, 26, 30, 34,
jaccard(), 48	37, 44, 47, 51, 55, 58, 60, 63, 64, 67,
jaccard.factor, 5, 10, 13, 16, 20, 23, 26, 30,	70, 73, 78, 81, 84, 89, 95, 97, 101,
33, 34, 37, 40, 44, 47, 48, 55, 58, 60,	104, 106, 109, 113, 115, 116, 118,
63, 65, 67, 70, 73, 78, 81, 84, 89, 94,	120, 122, 126, 130
95, 97, 101, 104, 106, 108, 109, 113,	mse (mse.numeric), 66
116, 118, 120, 122, 126, 130	mse(), 66
	mse.numeric, 5, 10, 13, 17, 20, 23, 26, 30, 34,
length, 3, 6–9, 13, 15, 18, 19, 22, 26, 28, 32,	<i>37</i> , <i>44</i> , <i>47</i> , <i>51</i> , <i>55</i> , <i>58</i> , <i>60</i> , <i>63</i> , <i>65</i> , <i>66</i> ,
36, 39, 43, 47, 50, 54, 57, 59, 61, 62,	70, 73, 78, 81, 84, 89, 95, 97, 101,
65, 67–69, 72, 78, 80, 83, 87, 88,	104, 106, 109, 113, 115, 116, 118,
91–93, 97, 100, 104, 106, 108, 111,	120, 122, 126, 130
115, 117, 119, 121, 124, 125, 129	
logical, 6, 9, 13, 28, 32, 36, 43, 50, 54, 72,	NA, 28, 32, 36, 43, 50, 72, 87, 100, 124
78, 83, 87, 91–93, 100, 108, 111, 124	nlr (nlr.factor), 68
logloss (logloss.factor), 53	nlr(), 81
logloss(), 53	nlr.factor, 5, 10, 13, 16, 17, 20, 23, 26, 30,
logloss.factor, 5, 10, 13, 16, 20, 23, 26, 30,	33, 34, 37, 40, 44, 47, 51, 55, 58, 60,
33, 34, 37, 40, 44, 47, 51, 53, 58, 60,	63, 65, 67, 68, 73, 78, 81, 84, 89, 94,

95, 97, 101, 104, 106, 108, 109, 113, 116, 118, 120, 122, 126, 130	precision.factor, 5, 10, 13, 16, 17, 20, 2. 26, 30, 33, 34, 37, 40, 44, 47, 51,
npv (npv.factor), 71	58, 60, 63, 65, 67, 70, 73, 78, 81,
npv(), 71	86, 94, 95, 97, 101, 104, 106, 108
npv. factor, 5, 10, 13, 16, 17, 20, 23, 26, 30,	109, 113, 116, 118, 120, 122, 126
33, 34, 37, 40, 44, 47, 51, 55, 58, 60,	130, 131
63, 65, 67, 70, 71, 78, 81, 84, 89, 94,	preorder, 7, 90, 92
95, 97, 101, 104, 106, 108, 109, 113,	presort, 7, 91, 91
116, 118, 120, 122, 126, 130, 131	prROC (prROC. factor), 92
NULL, 3, 9, 15, 19, 22, 28, 32, 36, 43, 47, 50,	prROC(), 92
54, 62, 69, 72, 76, 80, 83, 87, 88, 93,	prROC. factor, 5, 10, 13, 16, 17, 20, 23, 26
	•
100, 108, 111, 124, 125, 129	30, 33, 34, 37, 40, 44, 47, 51, 55,
numeric, 3, 6, 7, 9, 13, 15, 19, 22, 25, 26, 28,	60, 63, 65, 67, 70, 73, 78, 81, 84,
32, 36, 39, 43, 46, 47, 50, 53, 54, 57,	92, 97, 101, 104, 106, 108, 109, 1
59, 62, 64–67, 69, 72, 77, 78, 80, 83,	116, 118, 120, 122, 126, 130, 131
87, 88, 90–93, 96, 97, 100, 103–106,	
108, 111, 114–117, 119, 121, 125,	rae (rae.numeric), 96
129	rae(), 96
	rae.numeric, 5, 10, 13, 17, 20, 23, 26, 30,
obesity, 75	37, 44, 47, 51, 55, 58, 60, 63, 65,
openmp.off(openmp.on), 76	70, 73, 78, 81, 84, 89, 95, 96, 101
openmp.on, 76	104, 106, 109, 113, 115, 116, 118
openmp.threads(openmp.on), 76	120, 122, 126, 131
	range, <i>115</i>
phi (mcc.factor), 60	recall (recall.factor), 98
pinball (pinball.numeric), 77	recall(), <i>92</i> , <i>93</i>
pinball(), 77	recall.factor, 5, 10, 13, 16, 17, 20, 23, 20
pinball.numeric, 5, 10, 13, 17, 20, 23, 26,	30, 33, 34, 37, 40, 44, 47, 51, 55,
30, 34, 37, 44, 47, 51, 55, 58, 60, 63,	60, 63, 65, 67, 70, 73, 78, 81, 84,
65, 67, 70, 73, 77, 81, 84, 89, 95, 97,	94, 95, 97, 98, 104, 106, 108, 109
101, 104, 106, 109, 113, 115, 116,	113, 116, 118, 120, 122, 126, 130,
118, 120, 122, 126, 131	131
plr (plr.factor), 79	relative.entropy(entropy.matrix), 25
plr(), 70	rmse (rmse.numeric), 103
plr. factor, 5, 10, 13, 16, 17, 20, 23, 26, 30,	rmse(), 103
33, 34, 37, 40, 44, 47, 51, 55, 58, 60,	rmse.numeric, 5, 10, 13, 17, 20, 23, 26, 30,
63, 65, 67, 70, 73, 78, 79, 84, 89, 94,	34, 37, 44, 47, 51, 55, 58, 60, 63,
95, 97, 101, 104, 106, 108, 109, 113,	67, 70, 73, 78, 81, 84, 89, 95, 97,
116, 118, 120, 122, 126, 130, 131	101, 103, 106, 109, 113, 115, 116
ppv (precision. factor), 86	118, 120, 122, 126, 131
pr.auc (pr.auc.matrix), 82	rmsle (rmsle.numeric), 105
pr.auc.matrix, 5, 10, 13, 16, 17, 20, 23, 26,	
•	rmsle(), 105
30, 33, 34, 37, 40, 44, 47, 51, 55, 58,	rmsle.numeric, 5, 10, 13, 17, 20, 23, 26, 3
60, 63, 65, 67, 70, 73, 78, 81, 82, 89,	34, 37, 44, 47, 51, 55, 58, 60, 63,
94, 95, 97, 101, 104, 106, 108, 109,	67, 70, 73, 78, 81, 84, 89, 95, 97,
113, 116, 118, 120, 122, 126, 130,	101, 104, 105, 109, 113, 115, 116
	118, 120, 122, 126, 131
131	
precision (precision.factor), 86 precision(), 92, 93	ROC (ROC.factor), 110 ROC(), <i>110</i>

roc.auc(roc.auc.matrix), 107	tnr (specificity.factor), 123
roc.auc.matrix, 5, 10, 13, 16, 17, 20, 23, 26,	tpr (recall.factor), 98
30, 33, 34, 37, 40, 44, 47, 51, 55, 58,	tpr(), 110, 111
60, 63, 65, 67, 70, 73, 78, 81, 84, 89,	TRUE, 6, 9, 13, 15, 28, 32, 36, 43, 50, 54, 69,
94, 95, 97, 101, 104, 106, 107, 113,	72, 78, 80, 83, 87, 88, 91–93, 100,
116, 118, 120, 122, 126, 130, 131	108, 111, 124, 125
ROC. factor, 5, 10, 13, 16, 20, 23, 26, 30, 33,	tscore (jaccard.factor), 48
34, 37, 40, 44, 47, 51, 55, 58, 60, 63,	
65, 67, 70, 73, 78, 81, 84, 89, 94, 95,	weighted.accuracy(accuracy.factor), 3
97, 101, 104, 106, 108, 109, 110,	weighted.accuracy(), 3
115, 118, 120, 122, 126, 130	weighted.baccuracy(baccuracy.factor), 8
rrmse (rrmse.numeric), 114	weighted.baccuracy(), 8
rrmse(), 114	weighted.ccc(ccc.numeric), 12
rrmse.numeric, 5, 10, 13, 17, 20, 23, 26, 30,	weighted.ccc(), <i>12</i>
34, 37, 44, 47, 51, 55, 58, 60, 63, 65,	weighted.ckappa(ckappa.factor), 14
67, 70, 73, 78, 81, 84, 89, 95, 97,	weighted.ckappa(), <i>14</i>
101, 104, 106, 109, 113, 114, 118,	weighted.cmatrix(cmatrix.factor), 18
120, 122, 126, 131	weighted.csi(jaccard.factor),48
rrse (rrse.numeric), 116	weighted.dor(dor.factor),21
rrse(), 116	<pre>weighted.dor(), 21</pre>
rrse.numeric, 5, 10, 13, 17, 20, 23, 26, 30,	weighted.fallout(fpr.factor),41
34, 37, 44, 47, 51, 55, 58, 60, 63, 65,	weighted.fbeta(fbeta.factor), 27
67, 70, 73, 78, 81, 84, 89, 95, 97,	weighted.fbeta(), 27
101, 104, 106, 109, 113, 115, 116,	weighted.fdr(fdr.factor),31
116, 120, 122, 126, 131	weighted.fdr(), <i>31</i> , <i>35</i> , <i>86</i> , <i>98</i>
rsq (rsq. numeric), 118	weighted.fer(fer.factor),35
rsq.numeric, 5, 10, 13, 17, 20, 23, 26, 30, 34,	weighted.fpr(fpr.factor),41
37, 44, 47, 51, 55, 58, 60, 63, 65, 67,	weighted.fpr(), 42
70, 73, 78, 81, 84, 89, 95, 97, 101,	<pre>weighted.huberloss(huberloss.numeric),</pre>
	46
104, 106, 109, 113, 115, 116, 118, 118, 122, 126, 131	weighted.huberloss(),46
110, 122, 120, 131	weighted.jaccard(jaccard.factor),48
	weighted.jaccard(), 48
selectivity (specificity.factor), 123	weighted.logloss(logloss.factor), 53
sensitivity (recall.factor), 98	weighted.logloss(), 53
smape (smape.numeric), 121	weighted.mae(mae.numeric), 56
smape(), <i>121</i>	weighted.mae(), 57
smape.numeric, 5, 10, 13, 17, 20, 23, 26, 30,	weighted.mape(mape.numeric), 58
34, 37, 44, 47, 51, 55, 58, 60, 63, 65,	<pre>weighted.mape(), 59</pre>
67, 70, 73, 78, 81, 84, 89, 95, 97,	weighted.mcc(mcc.factor), 60
101, 104, 106, 109, 113, 115, 116,	weighted.mcc(), 61
<i>118, 120,</i> 121 <i>, 126, 131</i>	weighted.mpe(mpe.numeric), 64
specificity (specificity.factor), 123	weighted.mpe(), 64
specificity.factor, 5, 10, 13, 16, 17, 20,	weighted.mse (mse.numeric), 66
23, 26, 30, 33, 34, 37, 40, 44, 47, 51,	weighted.mse(), 66
55, 58, 60, 63, 65, 67, 70, 73, 78, 81,	weighted.nlr(nlr.factor), 68
84, 89, 94, 95, 97, 101, 104, 106,	weighted.nlr(), 68
108, 109, 113, 116, 118, 120, 122,	weighted.npv(npv.factor), 71
123, <i>130</i> , <i>131</i>	<pre>weighted.npv(), 71</pre>

weighted.phi (mcc.factor), 60			
weighted.pinball(pinball.numeric), 77			
weighted.pinball(), 77			
weighted.plr(plr.factor), 79			
weighted.plr(), 79			
weighted.ppv (precision.factor), 86			
weighted.pr.auc(pr.auc.matrix), 82			
weighted.pr.auc(), 82			
weighted.precision(precision.factor),			
86			
weighted.prROC(prROC.factor), 92			
weighted.rae(rae.numeric),96			
weighted.rae(), 96			
weighted.recall(recall.factor), 98			
weighted.rmse(rmse.numeric), 103			
weighted.rmse(), 103			
weighted.rmsle(rmsle.numeric), 105			
weighted.rmsle(), 105			
weighted.ROC(ROC.factor), 110			
weighted.roc.auc(roc.auc.matrix), 107			
weighted.roc.auc(), 107			
weighted.rrmse(rrmse.numeric), 114			
weighted.rrmse(), 114			
weighted.rrse(rrse.numeric), 116			
weighted.rrse(), 116			
weighted.rsq(rsq.numeric), 118			
weighted.rsq(), 118			
weighted.selectivity			
(specificity.factor), 123			
weighted.sensitivity(recall.factor), 98			
weighted.smape(smape.numeric), 121			
weighted.smape(), 121			
weighted.specificity			
(specificity.factor), 123			
weighted.specificity(), 123			
weighted.tnr (specificity.factor), 123			
weighted.thr (recall.factor), 98			
weighted.tscore (jaccard.factor), 48			
weighted.zerooneloss			
(zerooneloss.factor), 129			
weighted.zerooneloss(), 129			
wine_quality, 127			
wine_quality, 127			
zerooneloss (zerooneloss.factor), 129			
zerooneloss(), 129			
zerooneloss.factor, <i>5</i> , <i>10</i> , <i>13</i> , <i>16</i> , <i>17</i> , <i>20</i> ,			
23, 26, 30, 33, 34, 37, 40, 44, 47, 51,			
55, 58, 60, 63, 65, 67, 70, 73, 78, 81,			
84, 89, 94, 95, 97, 101, 104, 106,			

108, 109, 113, 116, 118, 120, 122, 126, 129