

Package ‘SLmetrics’

June 21, 2025

Title Machine Learning Performance Evaluation on Steroids

Version 0.3-4

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.

SystemRequirements C++17

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

LinkingTo Rcpp, RcppArmadillo

Suggests knitr, reticulate, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports grDevices, lattice, Rcpp

Depends R (>= 4.0.0)

URL <https://slmetrics-docs.gitbook.io/v1>,
<https://github.com/serkor1/SLmetrics>

BugReports <https://github.com/serkor1/SLmetrics/issues>

LazyData true

VignetteBuilder knitr

NeedsCompilation yes

Author Serkan Korkmaz [cre, aut, cph] (ORCID:
<<https://orcid.org/0000-0002-5052-0982>>)

Maintainer Serkan Korkmaz <serkor1@duck.com>

Repository CRAN

Date/Publication 2025-06-21 20:40:02 UTC

Contents

accuracy	5
accuracy.cmatrix	8
accuracy.factor	10
auc.pr.curve	12
auc.pr.curve.factor	15
auc.roc.curve	18
auc.roc.curve.factor	21
auc.xy	24
auc.xy.numeric	25
baccuracy	25
baccuracy.cmatrix	28
baccuracy.factor	30
banknote	33
brier.score	34
brier.score.matrix	36
ccc	38
ccc.numeric	39
ckappa	41
ckappa.cmatrix	44
ckappa.factor	46
cmatrix	48
cmatrix.factor	51
cross.entropy	53
cross.entropy.matrix	56
deviance.gamma	58
deviance.gamma.numeric	59
deviance.poisson	61
deviance.poisson.numeric	63
deviance.tweedie	65
deviance.tweedie.numeric	67
dor	69
dor.cmatrix	71
dor.factor	73
fbeta	76
fbeta.cmatrix	79
fbeta.factor	81
fdr	84
fdr.cmatrix	87
fdr.factor	89
fer	92
fer.cmatrix	94
fer.factor	97
fmi	100
fmi.cmatrix	102
fmi.factor	104
fpr	107

fpr.cmatrix	109
fpr.factor	112
gmse	115
gmse.numeric	117
hammingloss	118
hammingloss.cmatrix	121
hammingloss.factor	123
huberloss	125
huberloss.numeric	127
jaccard	129
jaccard.cmatrix	132
jaccard.factor	134
logloss	137
logloss.factor	140
logloss.integer	142
maape	144
maape.numeric	146
mae	148
mae.numeric	149
mape	151
mape.numeric	153
mcc	154
mcc.cmatrix	157
mcc.factor	159
mpe	162
mpe.numeric	163
mse	165
mse.numeric	167
nlr	169
nlr.cmatrix	171
nlr.factor	174
npv	176
npv.cmatrix	179
npv.factor	181
obesity	184
OpenMP	185
pinball	186
pinball.numeric	188
plr	190
plr.cmatrix	192
plr.factor	195
pr.curve	197
pr.curve.factor	200
precision	203
precision.cmatrix	205
precision.factor	208
preorder	211
preorder.matrix	212

presort	213
presort.matrix	214
rae	215
rae.numeric	216
recall	218
recall.cmatrix	221
recall.factor	224
relative.entropy	226
relative.entropy.matrix	229
rmse	231
rmse.numeric	232
rmsle	234
rmsle.numeric	236
roc.curve	238
roc.curve.factor	240
rrmse	243
rrmse.numeric	245
rrse	247
rrse.numeric	249
rsq	250
rsq.numeric	252
shannon.entropy	254
shannon.entropy.matrix	256
smape	258
smape.numeric	260
specificity	262
specificity.cmatrix	265
specificity.factor	267
weighted.accuracy.factor	270
weighted.auc.pr.curve.factor	273
weighted.auc.roc.curve.factor	276
weighted.baccracy.factor	279
weighted.brier.score.matrix	281
weighted.ccc.numeric	283
weighted.ckappa.factor	285
weighted.cmatrix.factor	288
weighted.deviance.gamma.numeric	290
weighted.deviance.poisson.numeric	292
weighted.deviance.tweedie.numeric	294
weighted.dor.factor	296
weighted.fbeta.factor	299
weighted.fdr.factor	302
weighted.fer.factor	305
weighted.fmi.factor	307
weighted.fpr.factor	310
weighted.gmse.numeric	313
weighted.hammingloss.factor	314
weighted.huberloss.numeric	317

weighted.jaccard.factor	319
weighted.logloss.factor	322
weighted.logloss.integer	324
weighted.maape.numeric	326
weighted.mae.numeric	328
weighted.mape.numeric	330
weighted.mcc.factor	332
weighted.mpe.numeric	334
weighted.mse.numeric	336
weighted.nlr.factor	338
weighted.npv.factor	340
weighted.pinball.numeric	343
weighted.plr.factor	345
weighted.pr.curve.factor	348
weighted.precision.factor	351
weighted.rae.numeric	354
weighted.recall.factor	355
weighted.rmse.numeric	358
weighted.rmsle.numeric	360
weighted.roc.curve.factor	362
weighted.rrmse.numeric	365
weighted.rrse.numeric	367
weighted.rsq.numeric	369
weighted.smape.numeric	371
weighted.specificity.factor	372
weighted.zeroone.loss.factor	375
wine.quality	378
zeroone.loss	379
zeroone.loss.cmatrix	381
zeroone.loss.factor	384

Index	387
--------------	------------

accuracy	<i>Accuracy</i>
----------	-----------------

Description

A generic S3 function to compute the *accuracy* score for a classification model. This function dispatches to S3 methods in `accuracy()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `accuracy()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `accuracy()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_accuracy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  accuracy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate accuracy
## via S3 dispatching
accuracy(confusion_matrix)

## additional performance metrics
## below
```

The `accuracy.factor()` method calls `cmatrix()` internally, so explicitly invoking `accuracy.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Accuracy
accuracy(...)

## Generic S3 method
## for weighted Accuracy
weighted.accuracy(...)
```

Arguments

... Arguments passed on to `accuracy.factor`, `weighted.accuracy.factor`, `accuracy.cmatrix`
 actual, predicted A pair of `<integer>` or `<factor>` vectors of length n , and k
 levels.
 w A `<double>` vector of sample weights.
 x A confusion matrix created `cmatrix()`.

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::accuracy(
  actual    = actual_classes,
  predicted = predicted_classes
)
```

accuracy.cmatrix

*Accuracy***Description**

A generic S3 function to compute the *accuracy* score for a classification model. This function dispatches to S3 methods in `accuracy()` and performs no input validation. If you supply `NA` values or vectors of unequal *length* (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `accuracy()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched *length*) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `accuracy()` in a "safe" validator that checks for `NA` values and matching *length*, for example:

```
safe_accuracy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  accuracy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate accuracy
## via S3 dispatching
accuracy(confusion_matrix)

## additional performance metrics
## below
```

The `accuracy.factor()` method calls `cmatrix()` internally, so explicitly invoking `accuracy.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

x A confusion matrix created `cmatrix()`.
 ... Arguments passed into other methods.

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)
```

```

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual    = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::accuracy(confusion_matrix)

```

accuracy.factor	<i>Accuracy</i>
-----------------	-----------------

Description

A generic S3 function to compute the *accuracy* score for a classification model. This function dispatches to S3 methods in `accuracy()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `accuracy()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `accuracy()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_accuracy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  accuracy(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate accuracy
## via S3 dispatching
accuracy(confusion_matrix)
```

```
## additional performance metrics
## below
```

The `accuracy.factor()` method calls `cmatrix()` internally, so explicitly invoking `accuracy.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

```
actual, predicted      A pair of <integer> or <factor> vectors of length  $n$ , and  $k$  levels.
...                   Arguments passed into other methods.
```

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::accuracy(
  actual = actual_classes,
  predicted = predicted_classes
)
```

auc.pr.curve

Area under the Precision Recall Curve

Description

A generic S3 function to compute the *area under the precision recall curve* score for a classification model. This function dispatches to S3 methods in `auc.pr.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `auc.pr.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `auc.pr.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_auc.pr.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
}
```

```
    auc.pr.curve(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Visualizing area under the precision recall curve:

Use `pr.curve()` to construct the `data.frame` and use `plot` to visualize the area under the curve.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```
## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate area under the precision recall curve
auc.pr.curve(actual, response, indices = indices)
```

Usage

```
## Generic S3 method
## for Area under the Precision Recall Curve
auc.pr.curve(...)

## Generic S3 method for
## unweighted area under the
## Precision Recall Curve
auc.pr.curve(...)

## Generic S3 method
## for weighted Area under the Precision Recall Curve
weighted.auc.pr.curve(...)
```

Arguments

...	Arguments passed on to <code>auc.pr.curve.factor</code> , <code>weighted.auc.pr.curve.factor</code>
actual	A vector length n , and k levels. Can be of integer or factor .
response	A $n \times k$ <double> -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 <code>actual</code> , the second column to the second factor level, and so on.
method	A <double> 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.
indices	An optional $n \times k$ matrix of <integer> values of sorted response probability indices.

estimator An [integer](#)-value of [length](#) 1 (default: 0).

- 0 - a named [double](#)-vector of [length](#) k (class-wise)
- 1 - a [double](#) value (Micro averaged metric)
- 2 - a [double](#) value (Macro averaged metric)

w A [double](#) vector of sample weights.

Value

If estimator is given as

- 0: a named [double](#)-vector of [length](#) k
- 1: a [double](#) value (Micro averaged metric)
- 2: a [double](#) value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
```

```

        size = 1e2,
        replace = TRUE,
        prob = c(0.7, 0.3)
      )
    )

    response_probabilities <- ifelse(
      actual_classes == "Kebab",
      rbeta(sum(actual_classes == "Kebab"), 2, 5),
      rbeta(sum(actual_classes == "Falafel"), 5, 2)
    )

    ## Construct response
    ## matrix
    probability_matrix <- cbind(
      response_probabilities,
      1 - response_probabilities
    )

    ## Calculate area under the precision recall curve

    SLmetrics::auc.pr.curve(
      actual = actual_classes,
      response = probability_matrix
    )

```

auc.pr.curve.factor *Area under the Precision Recall Curve*

Description

A generic S3 function to compute the *area under the precision recall curve* score for a classification model. This function dispatches to S3 methods in [auc.pr.curve\(\)](#) and performs no input validation. If you supply **NA** values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because [auc.pr.curve\(\)](#) operates on raw pointers, pointer-level faults (e.g. from **NA** or mismatched **length**) occur before any R-level error handling. Wrapping calls in [try\(\)](#) or [tryCatch\(\)](#) will *not* prevent R-session crashes.

To guard against this, wrap [auc.pr.curve\(\)](#) in a "safe" validator that checks for **NA** values and matching **length**, for example:

```

safe_auc.pr.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  auc.pr.curve(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Visualizing area under the precision recall curve:

Use `pr.curve()` to construct the `data.frame` and use `plot` to visualize the area under the curve.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```
## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate area under the precision recall curve
auc.pr.curve(actual, response, indices = indices)
```

Usage

```
## S3 method for class 'factor'
auc.pr.curve(
  actual,
  response,
  estimator = 0L,
  method = 0L,
  indices = NULL,
  ...
)
```

Arguments

actual	A vector <code>length</code> n , and k levels. Can be of <code>integer</code> or <code>factor</code> .
response	A $n \times k$ <code><double></code> -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 <code>actual</code> , the second column to the second factor level, and so on.
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
method	A <code><double></code> 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.
indices	An optional $n \times k$ matrix of <code><integer></code> values of sorted response probability indices.
...	Arguments passed into other methods.

Value

If estimator is given as

- 0: a named `<double>`-vector of `length` k
- 1: a `<double>` value (Micro averaged metric)
- 2: a `<double>` value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
```

```

    rbeta(sum(actual_classes == "Kebab"), 2, 5),
    rbeta(sum(actual_classes == "Falafel"), 5, 2)
  )

  ## Construct response
  ## matrix
  probability_matrix <- cbind(
    response_probabilities,
    1 - response_probabilities
  )

  ## Evaluate performance

  SLmetrics::auc.pr.curve(
    actual   = actual_classes,
    response = probability_matrix
  )

```

auc.roc.curve

Area under the Receiver Operator Characteristics Curve

Description

A generic S3 function to compute the *area under the receiver operator characteristics curve* score for a classification model. This function dispatches to S3 methods in `auc.roc.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `auc.roc.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `auc.roc.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_auc.roc.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  auc.roc.curve(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Visualizing area under the receiver operator characteristics curve:

Use `roc.curve()` to construct the `data.frame` and use `plot` to visualize the area under the curve.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```
## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate area under the receiver operator characteristics curve
auc.roc.curve(actual, response, indices = indices)
```

Usage

```
## Generic S3 method
## for Area under the Receiver Operator Characteristics Curve
auc.roc.curve(...)

## Generic S3 method for
## unweighted area under the
## Receiver Operator Characteristics
## Curve
auc.roc.curve(...)

## Generic S3 method
## for weighted Area under the Receiver Operator Characteristics Curve
weighted.auc.roc.curve(...)
```

Arguments

...	Arguments passed on to <code>auc.roc.curve.factor</code> , <code>weighted.auc.roc.curve.factor</code>
actual	A vector <code>length</code> n , and k levels. Can be of <code>integer</code> or <code>factor</code> .
response	A $n \times k$ <code><double></code> -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 <code>actual</code> , the second column to the second factor level, and so on.
method	A <code><double></code> 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.
indices	An optional $n \times k$ matrix of <code><integer></code> values of sorted response probability indices.
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
w	A <code><double></code> vector of sample weights.

Value

If estimator is given as

- 0: a named `<double>`-vector of `length` k
- 1: a `<double>` value (Micro averaged metric)
- 2: a `<double>` value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
```

```

    rbeta(sum(actual_classes == "Kebab"), 2, 5),
    rbeta(sum(actual_classes == "Falafel"), 5, 2)
  )

  ## Construct response
  ## matrix
  probability_matrix <- cbind(
    response_probabilities,
    1 - response_probabilities
  )

  ## Calculate area under the receiver operator characteristics curve

  SLmetrics::auc.roc.curve(
    actual = actual_classes,
    response = probability_matrix
  )

```

auc.roc.curve.factor *Area under the Receiver Operator Characteristics Curve*

Description

A generic S3 function to compute the *area under the receiver operator characteristics curve* score for a classification model. This function dispatches to S3 methods in `auc.roc.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `auc.roc.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `auc.roc.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_auc.roc.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  auc.roc.curve(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Visualizing area under the receiver operator characteristics curve:

Use `roc.curve()` to construct the `data.frame` and use `plot` to visualize the area under the curve.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```
## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate area under the receiver operator characteristics curve
auc.roc.curve(actual, response, indices = indices)
```

Usage

```
## S3 method for class 'factor'
auc.roc.curve(
  actual,
  response,
  estimator = 0L,
  method = 0L,
  indices = NULL,
  ...
)
```

Arguments

<code>actual</code>	A vector length n , and k levels. Can be of integer or factor .
<code>response</code>	A $n \times k$ <double> -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 <code>actual</code> , the second column to the second factor level, and so on.
<code>estimator</code>	An <integer> -value of length 1 (default: 0). • 0 - a named <double>-vector of length k (class-wise) • 1 - a <double> value (Micro averaged metric) • 2 - a <double> value (Macro averaged metric)
<code>method</code>	A <double> 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.
<code>indices</code>	An optional $n \times k$ matrix of <integer> values of sorted response probability indices.
<code>...</code>	Arguments passed into other methods.

Value

If estimator is given as

- 0: a named **<double>**-vector of **length** k
- 1: a **<double>** value (Micro averaged metric)
- 2: a **<double>** value (Macro averaged metric)

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
  rbeta(sum(actual_classes == "Kebab"), 2, 5),
  rbeta(sum(actual_classes == "Falafel"), 5, 2)
)

## Construct response
## matrix
probability_matrix <- cbind(
  response_probabilities,
  1 - response_probabilities
)
```

```

)

## Evaluate performance

SLmetrics::auc.roc.curve(
  actual   = actual_classes,
  response = probability_matrix
)

```

auc.xy

Area under the curve

Description

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

Usage

```
## Generic S3 method
auc.xy(...)
```

Arguments

... Arguments passed on to `auc.xy.numeric`

y, x A pair of `<double>` vectors of `length` n .

method A `<integer>` 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.

Value

A `<double>` value.

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 $x_0, x_1, \dots, x_n$ (with $a = x_0 < x_1 < \dots < x_n = b$) and corresponding function values $f(x_0), f(x_1), \dots, f(x_n)$, 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 $x_0, x_1, \dots, x_n$, the rectangular approximation A_S can be written as:

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

auc.xy.numeric	<i>Area under the curve</i>
----------------	-----------------------------

Description

Area under the curve

Usage

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

Arguments

y, x	A pair of <double> vectors of length n .
method	A <integer> 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.

baccuracy	<i>Balanced Accuracy</i>
-----------	--------------------------

Description

A generic S3 function to compute the *balanced accuracy* score for a classification model. This function dispatches to S3 methods in **baccuracy()** and performs no input validation. If you supply **NA** values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because **baccuracy()** operates on raw pointers, pointer-level faults (e.g. from **NA** or mismatched **length**) occur before any R-level error handling. Wrapping calls in **try()** or **tryCatch()** will *not* prevent R-session crashes.

To guard against this, wrap **baccuracy()** in a "safe" validator that checks for **NA** values and matching **length**, for example:

```
safe_baccuracy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  baccuracy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate balanced accuracy
## via S3 dispatching
baccuracy(confusion_matrix)

## additional performance metrics
## below
```

The `baccuracy.factor()` method calls `cmatrix()` internally, so explicitly invoking `baccuracy.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Balanced Accuracy
baccuracy(...)

## Generic S3 method
## for weighted Balanced Accuracy
weighted.baccuracy(...)
```

Arguments

...	Arguments passed on to <code>baccuracy.factor</code> , <code>weighted.baccuracy.factor</code> , <code>baccuracy.cmatrix</code>
adjust	A <code><logical></code> value (default: <code>FALSE</code>). If <code>TRUE</code> the metric is adjusted for random chance $\frac{1}{k}$.
actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of length n , and k levels.

`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))`.

`w` A [double](#) vector of sample weights.

`x` A confusion matrix created [cmatrix\(\)](#).

Value

A [double](#)-value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)
```

```

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::baccuracy(
  actual = actual_classes,
  predicted = predicted_classes
)

```

baccuracy.cmatrix	<i>Balanced Accuracy</i>
-------------------	--------------------------

Description

A generic S3 function to compute the *balanced accuracy* score for a classification model. This function dispatches to S3 methods in `baccuracy()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `baccuracy()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `baccuracy()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_baccuracy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  baccuracy(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```

## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate balanced accuracy

```

```
## via S3 dispatching
baccuracy(confusion_matrix)
```

```
## additional performance metrics
## below
```

The `baccuracy.factor()` method calls `cmatrix()` internally, so explicitly invoking `baccuracy.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
baccuracy(x, adjust = FALSE, na.rm = TRUE, ...)
```

Arguments

<code>x</code>	A confusion matrix created <code>cmatrix()</code> .
<code>adjust</code>	A <code><logical></code> value (default: <code>FALSE</code>). If <code>TRUE</code> the metric is adjusted for random chance $\frac{1}{k}$.
<code>na.rm</code>	A <code><logical></code> value of length 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
<code>...</code>	Arguments passed into other methods.

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::baccuracy(confusion_matrix)
```

baccuracy.factor

Balanced Accuracy

Description

A generic S3 function to compute the *balanced accuracy* score for a classification model. This function dispatches to S3 methods in `baccuracy()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `baccuracy()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `baccuracy()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_baccuracy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  baccuracy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate balanced accuracy
## via S3 dispatching
baccuracy(confusion_matrix)

## additional performance metrics
## below
```

The `baccuracy.factor()` method calls `cmatrix()` internally, so explicitly invoking `baccuracy.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
baccuracy(actual, predicted, adjust = FALSE, na.rm = TRUE, ...)
```

Arguments

`actual, predicted`

A pair of `<integer>` or `<factor>` vectors of `length` n , and k levels.

`adjust`

A `<logical>` value (default: `FALSE`). If `TRUE` the metric is adjusted for random chance $\frac{1}{k}$.

`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))`.

`...` Arguments passed into other methods.

Value

A [double](#)-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
```

```

    x = sample(x = classes, size = 1e3, replace = TRUE),
    levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::baccuracy(
  actual    = actual_classes,
  predicted = predicted_classes
)

```

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.

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.

Usage

```
data(banknote)
```

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.

References

Gillich, Eugen & Lohweg, Volker. (2010). Banknote Authentication.

brier.score

*Brier Score***Description**

A generic S3 function to compute the *brier score* score for a classification model. This function dispatches to S3 methods in `brier.score()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `brier.score()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `brier.score()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_brier.score <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  brier.score(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Brier Score
brier.score(...)

## Generic S3 method
## for weighted Brier Score
weighted.brier.score(...)
```

Arguments

- ... Arguments passed on to `brier.score.matrix`, `weighted.brier.score.matrix`
- ok A `<double>` indicator matrix with n samples and k classes.
- qk A $n \times k$ `<double>`-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.
- w A `<double>` vector of sample weights.

Value

A `<double>`-value

References

Gneiting, Tilmann, and Adrian E. Raftery. "Strictly proper scoring rules, prediction, and estimation." *Journal of the American statistical Association* 102.477 (2007): 359-378.

James, Gareth, et al. *An introduction to statistical learning*. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *the Journal of machine Learning research* 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

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

## The general setup
## with 3 classes
n_obs    <- 10
n_classes <- 3

## Generate indicator matrix
## with observed outcome (ok) and
## its predicted probability matrix (qk)
ok <- diag(n_classes)[ sample.int(n_classes, n_obs, TRUE), ]
qk <- matrix(runif(n_obs * n_classes), n_obs, n_classes)
qk <- qk / rowSums(qk)

## Evaluate performance
SLmetrics::brier.score(
  ok = ok,
  qk = qk
)
```

brier.score.matrix *Brier Score*

Description

A generic S3 function to compute the *brier score* score for a classification model. This function dispatches to S3 methods in `brier.score()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `brier.score()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `brier.score()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_brier.score <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  brier.score(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## S3 method for class 'matrix'
brier.score(ok, qk, ...)
```

Arguments

<code>ok</code>	A <code><double></code> indicator matrix with n samples and k classes.
<code>qk</code>	A $n \times k$ <code><double></code> -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 <code>actual</code> , the second column to the second factor level, and so on.
<code>...</code>	Arguments passed into other methods.

Value

A `<double>`-value

References

Gneiting, Tilmann, and Adrian E. Raftery. "Strictly proper scoring rules, prediction, and estimation." *Journal of the American statistical Association* 102.477 (2007): 359-378.

James, Gareth, et al. *An introduction to statistical learning*. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *the Journal of machine Learning research* 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

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

## The general setup
## with 3 classes
n_obs    <- 10
n_classes <- 3

## Generate indicator matrix
## with observed outcome (ok) and
## its predicted probability matrix (qk)
ok <- diag(n_classes)[ sample.int(n_classes, n_obs, TRUE), ]
qk <- matrix(runif(n_obs * n_classes), n_obs, n_classes)
qk <- qk / rowSums(qk)

## Evaluate performance
SLmetrics::brier.score(
  ok = ok,
  qk = qk
)
```

ccc

*Concordance Correlation Coefficient***Description**

A generic S3 function to compute the *concordance correlation coefficient* score for a regression model. This function dispatches to S3 methods in `ccc()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `ccc()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `ccc()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_ccc <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  ccc(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Concordance Correlation Coefficient
ccc(...)

## Generic S3 method
## for weighted Concordance Correlation Coefficient
weighted.ccc(...)
```

Arguments

... Arguments passed on to `ccc.numeric`, `weighted.ccc.numeric`

actual, predicted A pair of `<double>` vectors of `length` `n`.

correction A `<logical>` vector of `length` 1 (default: `FALSE`). If `TRUE` the variance and covariance will be adjusted with $\frac{1-n}{n}$

w A `<double>` vector of sample weights.

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone.loss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::ccc(
  actual    = actual_values,
  predicted = predicted_values
)
```

Description

A generic S3 function to compute the *concordance correlation coefficient* score for a regression model. This function dispatches to S3 methods in `ccc()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `ccc()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `ccc()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_ccc <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  ccc(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## S3 method for class 'numeric'
ccc(actual, predicted, correction = FALSE, ...)
```

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length</code> n .
<code>correction</code>	A <code><logical></code> vector of <code>length</code> 1 (default: <code>FALSE</code>). If <code>TRUE</code> the variance and covariance will be adjusted with $\frac{1-n}{n}$
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone.loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::ccc(
  actual = actual_values,
  predicted = predicted_values
)
```

ckappa

Cohen's κ -Statistic

Description

A generic S3 function to compute the *cohen's κ -statistic* score for a classification model. This function dispatches to S3 methods in `ckappa()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `ckappa()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `ckappa()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_ckappa <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  ckappa(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate cohen's \eqn{\kappa}-statistic
## via S3 dispatching
ckappa(confusion_matrix)

## additional performance metrics
## below
```

The `ckappa.factor()` method calls `cmatrix()` internally, so explicitly invoking `ckappa.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Cohen's \eqn{\kappa}-Statistic
ckappa(...)

## Generic S3 method
## for weighted Cohen's \eqn{\kappa}-Statistic
weighted.ckappa(...)
```

Arguments

```
...      Arguments passed on to ckappa.factor, weighted.ckappa.factor, ckappa.cmatrix
beta A <double> value of length 1 (default: 0). If  $\beta \neq 0$  the off-diagonals of
      the confusion matrix are penalized with a factor of  $(y_+ - y_{i,-})^\beta$ .
actual, predicted A pair of <integer> or <factor> vectors of length  $n$ , and  $k$ 
      levels.
w A <double> vector of sample weights.
x A confusion matrix created cmatrix().
```

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::ckappa(
  actual = actual_classes,
  predicted = predicted_classes
```

)

ckappa.cmatrix

Cohen's κ -Statistic

Description

A generic S3 function to compute the *cohen's κ -statistic* score for a classification model. This function dispatches to S3 methods in `ckappa()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `ckappa()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `ckappa()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_ckappa <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  ckappa(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate cohen's \eqn{\kappa}-statistic
## via S3 dispatching
ckappa(confusion_matrix)

## additional performance metrics
## below
```

The `ckappa.factor()` method calls `cmatrix()` internally, so explicitly invoking `ckappa.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

x	A confusion matrix created <code>cmatrix()</code> .
beta	A <double> 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.

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
```

```

x = sample(x = classes, size = 1e3, replace = TRUE),
levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual    = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::ckappa(confusion_matrix)

```

ckappa.factor

Cohen's κ -Statistic

Description

A generic S3 function to compute the *cohen's κ -statistic* score for a classification model. This function dispatches to S3 methods in `ckappa()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `ckappa()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `ckappa()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_ckappa <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  ckappa(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate cohen's  $\kappa$ -statistic
## via S3 dispatching
ckappa(confusion_matrix)

## additional performance metrics
## below
```

The `ckappa.factor()` method calls `cmatrix()` internally, so explicitly invoking `ckappa.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of length n , and k levels.
beta	A <code><double></code> 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.

Value

A `<double>`-value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::ckappa(
  actual    = actual_classes,
  predicted = predicted_classes
)
```

cmatrix

Confusion Matrix

Description

A generic S3 function to compute the *confusion matrix* for a classification model. This function dispatches to S3 methods in `cmatrix()` and performs no input validation. If you supply `NA` values

or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `cmatrix()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `cmatrix()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_cmatrix <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  cmatrix(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

The workhorse:

`cmatrix()` is the main function for classification metrics with `cmatrix` S3 dispatch. These functions internally calls `cmatrix()`, so there is a significant gain in computing the confusion matrix first, and then pass it onto the metrics. For example:

```
## Compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## Evaluate accuracy
## via S3 dispatching
accuracy(confusion_matrix)

## Evaluate recall
## via S3 dispatching
recall(confusion_matrix)
```

Usage

```
## Generic S3 method
## for Confusion Matrix
cmatrix(...)

## Generic S3 method
## for weighted Confusion Matrix
weighted.cmatrix(...)
```

Arguments

... Arguments passed on to `cmatrix.factor`, `weighted.cmatrix.factor`

actual, predicted A pair of [integer](#) or [factor](#) vectors of [length](#) n , and k levels.

w A [double](#) vector of sample weights.

Value

A named $k \times k$ [matrix](#)

Dimensions

There is no robust defensive measure against misspecifying the confusion matrix. If the arguments are passed correctly, the resulting confusion matrix is on the form:

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

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
```

```
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Compute the confusion
## matrix
SLmetrics::cmatrix(
  actual    = actual_classes,
  predicted = predicted_classes
)
```

cmatrix.factor

Confusion Matrix

Description

A generic S3 function to compute the *confusion matrix* for a classification model. This function dispatches to S3 methods in `cmatrix()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `cmatrix()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `cmatrix()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_cmatrix <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  cmatrix(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

The workhorse:

`cmatrix()` is the main function for classification metrics with `cmatrix` S3 dispatch. These functions internally call `cmatrix()`, so there is a significant gain in computing the confusion matrix first, and then pass it onto the metrics. For example:

```
## Compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## Evaluate accuracy
## via S3 dispatching
accuracy(confusion_matrix)

## Evaluate recall
## via S3 dispatching
recall(confusion_matrix)
```

Usage

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

Arguments

actual, predicted A pair of `<integer>` or `<factor>` vectors of `length` n , and k levels.

... Arguments passed into other methods.

Value

A named $k \times k$ `<matrix>`

Dimensions

There is no robust defensive measure against misspecifying the confusion matrix. If the arguments are passed correctly, the resulting confusion matrix is on the form:

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

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Compute confusion matrix
SLmetrics::cmatrix(
  actual    = actual_classes,
  predicted = predicted_classes
)
```

cross.entropy

Cross Entropy

Description

A generic S3 function to compute the *cross entropy* score for a classification model. This function dispatches to S3 methods in `cross.entropy()` and performs no input validation. If you supply `NA`

values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `cross.entropy()` operates on raw pointers, pointer-level faults (e.g. from **NA** or mismatched **length**) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `cross.entropy()` in a "safe" validator that checks for **NA** values and matching **length**, for example:

```
safe_cross.entropy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  cross.entropy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Cross Entropy
cross.entropy(...)
```

Arguments

... Arguments passed on to `cross.entropy.matrix`

pk, qk A pair of **<double>** matrices of **length** *n* of empirical probabilities *p* and estimated probabilities *q*.

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).

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.

Value

A **<double>** value or vector:

- A single **<double>** value (length 1) if `dim == 0`.
- A **<double>** vector with length equal to the **length** of columns if `dim == 1`.
- A **<double>** vector with length equal to the **length** of rows if `dim == 2`.

References

- MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.
- Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Entropy: [logloss\(\)](#), [relative.entropy\(\)](#), [shannon.entropy\(\)](#)

Examples

```
## generate valid probability
## distributions
rand.sum <- function(n) {
  x <- sort(runif( n-1 ))
  c(x,1) - c(0, x)
}

## empirical and
## predicted probabilities
set.seed(1903)
pk <- t(replicate(200,rand.sum(5)))
qk <- t(replicate(200,rand.sum(5)))

## entropy
cross.entropy(
  pk = pk,
  qk = qk
)
```

cross.entropy.matrix *Cross Entropy*

Description

A generic S3 function to compute the *cross entropy* score for a classification model. This function dispatches to S3 methods in `cross.entropy()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `cross.entropy()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `cross.entropy()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_cross.entropy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  cross.entropy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## S3 method for class 'matrix'
cross.entropy(pk, qk, dim = 0L, normalize = FALSE, ...)
```

Arguments

pk, qk	A pair of <double> matrices of <code>length</code> n of empirical probabilities p and estimated probabilities q .
dim	An <integer> value of <code>length</code> 1 (Default: 0). Defines the dimension along which to calculate the entropy (0: total, 1: row-wise, 2: column-wise).
normalize	A <logical>-value (default: <code>TRUE</code>). If <code>TRUE</code> , the mean cross-entropy across all observations is returned; otherwise, the sum of cross-entropies is returned.
...	Arguments passed into other methods.

Value

A [<double>](#) value or vector:

- A single [<double>](#) value (length 1) if `dim == 0`.
- A [<double>](#) vector with length equal to the [length](#) of columns if `dim == 1`.
- A [<double>](#) vector with length equal to the [length](#) of rows if `dim == 2`.

References

MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.

Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Entropy: [logloss\(\)](#), [relative.entropy\(\)](#), [shannon.entropy\(\)](#)

Examples

```
## generate valid probability
## distributions
rand.sum <- function(n) {
  x <- sort(runif( n-1 ))
  c(x,1) - c(0, x)
}

## empirical and
## predicted probabilities
set.seed(1903)
pk <- t(replicate(200,rand.sum(5)))
qk <- t(replicate(200,rand.sum(5)))

## entropy
cross.entropy(
  pk = pk,
```

```

    qk = qk
  )

```

deviance.gamma

Gamma Deviance

Description

A generic S3 function to compute the *gamma deviance* score for a regression model. This function dispatches to S3 methods in `deviance.gamma()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `deviance.gamma()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `deviance.gamma()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_deviance.gamma <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  deviance.gamma(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## Generic S3 method
## for Gamma Deviance
deviance.gamma(...)

## Generic S3 method
## for weighted Gamma Deviance
weighted.deviance.gamma(...)

```

Arguments

```

...      Arguments passed on to deviance.gamma.numeric, weighted.deviance.gamma.numeric
actual,predicted A pair of <double> vectors of length n.
w        A <double> vector of sample weights.

```

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::deviance.gamma(
  actual    = actual_values,
  predicted = predicted_values
)
```

Description

A generic S3 function to compute the *gamma deviance* score for a regression model. This function dispatches to S3 methods in `deviance.gamma()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `deviance.gamma()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `deviance.gamma()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_deviance.gamma <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  deviance.gamma(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length n</code> .
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::deviance.gamma(
  actual = actual_values,
  predicted = predicted_values
)
```

deviance.poisson	<i>Poisson Deviance</i>
------------------	-------------------------

Description

A generic S3 function to compute the *poisson deviance* score for a regression model. This function dispatches to S3 methods in `deviance.poisson()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `deviance.poisson()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `deviance.poisson()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_deviance.poisson <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  deviance.poisson(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Poisson Deviance
deviance.poisson(...)

## Generic S3 method
## for weighted Poisson Deviance
weighted.deviance.poisson(...)
```

Arguments

... Arguments passed on to `deviance.poisson.numeric`, `weighted.deviance.poisson.numeric`
 actual, predicted A pair of <double> vectors of length *n*.
 w A <double> vector of sample weights.

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::deviance.poisson(
  actual = actual_values,
  predicted = predicted_values
)
```

deviance.poisson.numeric

Poisson Deviance

Description

A generic S3 function to compute the *poisson deviance* score for a regression model. This function dispatches to S3 methods in `deviance.poisson()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `deviance.poisson()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `deviance.poisson()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_deviance.poisson <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  deviance.poisson(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted
 A pair of <double> vectors of length *n*.
 ... Arguments passed into other methods

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::deviance.poisson(
  actual    = actual_values,
  predicted = predicted_values
)
```

deviance.tweedie *Tweedie Deviance*

Description

A generic S3 function to compute the *tweedie deviance* score for a regression model. This function dispatches to S3 methods in `deviance.tweedie()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `deviance.tweedie()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `deviance.tweedie()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_deviance.tweedie <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  deviance.tweedie(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Tweedie Deviance
deviance.tweedie(...)

## Generic S3 method
## for weighted Tweedie Deviance
weighted.deviance.tweedie(...)
```

Arguments

```
...      Arguments passed on to deviance.tweedie.numeric, weighted.deviance.tweedie.numeric
actual,predicted A pair of <double> vectors of length n.
power      A <double> value, default = 2. Tweedie power parameter. Either power
           <= 0 or power >= 1.
           The higher power, the less weight is given to extreme deviations between
           actual and predicted values.
           • power < 0: Extreme stable distribution. Requires: predicted > 0.
```

- **power = 0:** Normal distribution, output corresponds to `mse()`, actual and predicted can be any real numbers.
- **power = 1:** Poisson distribution (`deviance.poisson()`). Requires: actual ≥ 0 and predicted > 0 .
- **1 < power < 2:** Compound Poisson distribution. Requires: actual ≥ 0 and predicted > 0 .
- **power = 2:** Gamma distribution (`deviance.gamma()`). Requires: actual > 0 and predicted > 0 .
- **power = 3:** Inverse Gaussian distribution. Requires: actual > 0 and predicted > 0 .
- **otherwise:** Positive stable distribution. Requires: actual > 0 and predicted > 0 .

w A `<double>` vector of sample weights.

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::deviance.tweedie(
```

```

    actual    = actual_values,
    predicted = predicted_values
  )

```

deviance.tweedie.numeric

Tweedie Deviance

Description

A generic S3 function to compute the *tweedie deviance* score for a regression model. This function dispatches to S3 methods in `deviance.tweedie()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `deviance.tweedie()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `deviance.tweedie()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_deviance.tweedie <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  deviance.tweedie(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## S3 method for class 'tweedie.numeric'
deviance(actual, predicted, power = 2, ...)

```

Arguments

`actual, predicted`

A pair of `<double>` vectors of `length` *n*.

`power`

A `<double>` value, default = 2. Tweedie power parameter. Either `power <= 0` or `power >= 1`.

The higher *power*, the less weight is given to extreme deviations between actual and predicted values.

- **power < 0:** Extreme stable distribution. Requires: `predicted > 0`.

- **power = 0:** Normal distribution, output corresponds to `mse()`, actual and predicted can be any real numbers.
- **power = 1:** Poisson distribution (`deviance.poisson()`). Requires: actual ≥ 0 and predicted > 0 .
- **1 < power < 2:** Compound Poisson distribution. Requires: actual ≥ 0 and predicted > 0 .
- **power = 2:** Gamma distribution (`deviance.gamma()`). Requires: actual > 0 and predicted > 0 .
- **power = 3:** Inverse Gaussian distribution. Requires: actual > 0 and predicted > 0 .
- **otherwise:** Positive stable distribution. Requires: actual > 0 and predicted > 0 .

... Arguments passed into other methods

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)
```

```
## Evaluate performance
SLmetrics::deviance.tweedie(
  actual    = actual_values,
  predicted = predicted_values
)
```

dor

Diagnostic Odds Ratio

Description

A generic S3 function to compute the *diagnostic odds ratio* score for a classification model. This function dispatches to S3 methods in `dor()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `dor()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `dor()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_dor <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  dor(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate diagnostic odds ratio
## via S3 dispatching
dor(confusion_matrix)

## additional performance metrics
## below
```

The `dor.factor()` method calls `cmatrix()` internally, so explicitly invoking `dor.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Diagnostic Odds Ratio
dor(...)
```

Arguments

```
...      Arguments passed on to dor.factor, weighted.dor.factor, dor.cmatrix
actual,predicted A pair of <integer> or <factor> vectors of length n, and k
           levels.
w A <double> vector of sample weights.
x A confusion matrix created cmatrix().
```

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::dor(
  actual = actual_classes,
  predicted = predicted_classes
)
```

dor.cmatrix

Diagnostic Odds Ratio

Description

A generic S3 function to compute the *diagnostic odds ratio* score for a classification model. This function dispatches to S3 methods in `dor()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `dor()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `dor()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_dor <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  dor(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate diagnostic odds ratio
## via S3 dispatching
dor(confusion_matrix)

## additional performance metrics
## below
```

The `dor.factor()` method calls `cmatrix()` internally, so explicitly invoking `dor.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

<code>x</code>	A confusion matrix created <code>cmatrix()</code> .
<code>...</code>	Arguments passed into other methods.

Value

A `<double>`-value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::dor(confusion_matrix)
```

Description

A generic S3 function to compute the *diagnostic odds ratio* score for a classification model. This function dispatches to S3 methods in `dor()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `dor()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `dor()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_dor <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  dor(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate diagnostic odds ratio
## via S3 dispatching
dor(confusion_matrix)

## additional performance metrics
## below
```

The `dor.factor()` method calls `cmatrix()` internally, so explicitly invoking `dor.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

actual, predicted
 A pair of <integer> or <factor> vectors of length n , and k levels.
 ... Arguments passed into other methods.

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)
```

```
## Evaluate performance
SLmetrics::dor(
  actual    = actual_classes,
  predicted = predicted_classes
)
```

fbeta

 f_{β}

Description

A generic S3 function to compute the f_{β} score for a classification model. This function dispatches to S3 methods in `fbeta()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fbeta()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fbeta()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fbeta <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fbeta(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate  $f_{\beta}$ 
## via S3 dispatching
fbeta(confusion_matrix)
```

```
## additional performance metrics
## below
```

The `fbeta.factor()` method calls `cmatrix()` internally, so explicitly invoking `fbeta.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for \eqn{f_{\beta}}
fbeta(...)

## Generic S3 method
## for weighted \eqn{f_{\beta}}
weighted.fbeta(...)
```

Arguments

... Arguments passed on to `fbeta.factor`, `weighted.fbeta.factor`, `fbeta.cmatrix`

beta A `<double>` vector of `length` 1 (default: 1).

actual, predicted A pair of `<integer>` or `<factor>` vectors of `length` n , and k levels.

estimator An `<integer>`-value of `length` 1 (default: 0).

- 0 - a named `<double>`-vector of `length` k (class-wise)
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

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))`.

w A `<double>` vector of sample weights.

x A confusion matrix created `cmatrix()`.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::fbeta(
  actual    = actual_classes,
  predicted = predicted_classes
)
```

fbeta.cmatrix	f_{β}
---------------	-------------

Description

A generic S3 function to compute the f_{β} score for a classification model. This function dispatches to S3 methods in `fbeta()` and performs no input validation. If you supply NA values or vectors of unequal length (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fbeta()` operates on raw pointers, pointer-level faults (e.g. from NA or mismatched length) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fbeta()` in a "safe" validator that checks for NA values and matching length, for example:

```
safe_fbeta <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fbeta(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate  $f_{\beta}$ 
## via S3 dispatching
fbeta(confusion_matrix)

## additional performance metrics
## below
```

The `fbeta.factor()` method calls `cmatrix()` internally, so explicitly invoking `fbeta.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
fbeta(x, beta = 1, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

x	A confusion matrix created <code>cmatrix()</code> .
beta	A <code><double></code> vector of <code>length</code> 1 (default: 1).
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`,

```
jaccard(), logloss(), maape(), mae(), mape(), mcc(), mpe(), mse(), nlr(), npv(), pinball(),
plr(), pr.curve(), precision(), rae(), recall(), relative.entropy(), rmse(), rmsle(),
roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(), zeroone loss()
```

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::fbeta(confusion_matrix)
```

fbeta.factor

 f_{β}

Description

A generic S3 function to compute the f_{β} score for a classification model. This function dispatches to S3 methods in `fbeta()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fbeta()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fbeta()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fbeta <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fbeta(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate \eqn{f_{\beta}}
## via S3 dispatching
fbeta(confusion_matrix)

## additional performance metrics
## below
```

The `fbeta.factor()` method calls `cmatrix()` internally, so explicitly invoking `fbeta.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
fbeta(actual, predicted, beta = 1, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

<code>actual, predicted</code>	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
<code>beta</code>	A <code><double></code> vector of <code>length</code> 1 (default: 1).
<code>estimator</code>	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)

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))`.

... Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named [<double>](#) vector of [length](#) k
- 1 - a [<double>](#) value (Micro averaged metric)
- 2 - a [<double>](#) value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
```

```

    levels = c("Kebab", "Falafel")
  )

  predicted_classes <- factor(
    x = sample(x = classes, size = 1e3, replace = TRUE),
    levels = c("Kebab", "Falafel")
  )

  ## Evaluate performance
  SLmetrics::fbeta(
    actual    = actual_classes,
    predicted = predicted_classes
  )

```

fdr

False Discovery Rate

Description

A generic S3 function to compute the *false discovery rate* score for a classification model. This function dispatches to S3 methods in `fdr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fdr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fdr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_fdr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fdr(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate false discovery rate
## via S3 dispatching
fdr(confusion_matrix)
```

```
## additional performance metrics
## below
```

The `fdr.factor()` method calls `cmatrix()` internally, so explicitly invoking `fdr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for False Discovery Rate
fdr(...)

## Generic S3 method
## for weighted False Discovery Rate
weighted.fdr(...)
```

Arguments

... Arguments passed on to `fdr.factor`, `weighted.fdr.factor`, `fdr.cmatrix`

actual, predicted A pair of `<integer>` or `<factor>` vectors of length n , and k levels.

estimator An `<integer>`-value of length 1 (default: 0).

- 0 - a named `<double>`-vector of length k (class-wise)
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

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))`.

w A `<double>` vector of sample weights.

x A confusion matrix created `cmatrix()`.

Value

If estimator is given as

- 0 - a named `<double>` vector of length k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::fdr(
  actual    = actual_classes,
  predicted = predicted_classes
)
```

Description

A generic S3 function to compute the *false discovery rate* score for a classification model. This function dispatches to S3 methods in `fdr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fdr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fdr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fdr <- function(x, y, ...) {  
  stopifnot(  
    !anyNA(x), !anyNA(y),  
    length(x) == length(y)  
  )  
  fdr(x, y, ...)  
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix  
confusion_matrix <- cmatrix(actual, predicted)  
  
## evaluate false discovery rate  
## via S3 dispatching  
fdr(confusion_matrix)  
  
## additional performance metrics  
## below
```

The `fdr.factor()` method calls `cmatrix()` internally, so explicitly invoking `fdr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
fdr(x, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

x	A confusion matrix created <code>cmatrix()</code> .
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::fdr(confusion_matrix)
```

fdr.factor

False Discovery Rate

Description

A generic S3 function to compute the *false discovery rate* score for a classification model. This function dispatches to S3 methods in `fdr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fdr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fdr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fdr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fdr(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate false discovery rate
## via S3 dispatching
fdr(confusion_matrix)

## additional performance metrics
## below
```

The `fdr.factor()` method calls `cmatrix()` internally, so explicitly invoking `fdr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
fdr(actual, predicted, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
```

```
SLmetrics::fdr(
  actual      = actual_classes,
  predicted   = predicted_classes
)
```

fer

False Omission Rate

Description

A generic S3 function to compute the *false omission rate* score for a classification model. This function dispatches to S3 methods in `fer()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fer()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fer()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fer <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fer(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate false omission rate
## via S3 dispatching
fer(confusion_matrix)

## additional performance metrics
## below
```

The `fer.factor()` method calls `cmatrix()` internally, so explicitly invoking `fer.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for False Omission Rate
fer(...)

## Generic S3 method
## for weighted False Omission Rate
weighted.fer(...)
```

Arguments

... Arguments passed on to `fer.factor`, `weighted.fer.factor`, `fer.cmatrix`

actual, predicted A pair of `<integer>` or `<factor>` vectors of length n , and k levels.

estimator An `<integer>`-value of length 1 (default: 0).

- 0 - a named `<double>`-vector of length k (class-wise)
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

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))`.

w A `<double>` vector of sample weights.

x A confusion matrix created `cmatrix()`.

Value

If estimator is given as

- 0 - a named `<double>` vector of length k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::fer(
  actual = actual_classes,
  predicted = predicted_classes
)
```

fer.cmatrix

False Omission Rate

Description

A generic S3 function to compute the *false omission rate* score for a classification model. This function dispatches to S3 methods in [fer\(\)](#) and performs no input validation. If you supply **NA** values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fer()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fer()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fer <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fer(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate false omission rate
## via S3 dispatching
fer(confusion_matrix)

## additional performance metrics
## below
```

The `fer.factor()` method calls `cmatrix()` internally, so explicitly invoking `fer.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
fer(x, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

- | | |
|-----------|--|
| x | A confusion matrix created <code>cmatrix()</code> . |
| estimator | An <code><integer></code> -value of <code>length</code> 1 (default: 0). <ul style="list-style-type: none"> • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric) |

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))`.

... Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named [<double>](#) vector of [length](#) k
- 1 - a [<double>](#) value (Micro averaged metric)
- 2 - a [<double>](#) value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
```

```

    levels = c("Kebab", "Falafel")
  )

  predicted_classes <- factor(
    x = sample(x = classes, size = 1e3, replace = TRUE),
    levels = c("Kebab", "Falafel")
  )

  ## Construct confusion
  ## matrix
  confusion_matrix <- SLmetrics::cmatrix(
    actual    = actual_classes,
    predicted = predicted_classes
  )

  ## Evaluate performance
  SLmetrics::fer(confusion_matrix)

```

fer.factor

False Omission Rate

Description

A generic S3 function to compute the *false omission rate* score for a classification model. This function dispatches to S3 methods in `fer()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fer()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fer()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_fer <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fer(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate false omission rate
## via S3 dispatching
fer(confusion_matrix)

## additional performance metrics
## below
```

The `fer.factor()` method calls `cmatrix()` internally, so explicitly invoking `fer.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
fer(actual, predicted, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::fer(
  actual = actual_classes,
  predicted = predicted_classes
)
```

Description

A generic S3 function to compute the *fowlkes mallows index* score for a classification model. This function dispatches to S3 methods in `fmi()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fmi()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fmi()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fmi <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fmi(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate fowlkes mallows index
## via S3 dispatching
fmi(confusion_matrix)

## additional performance metrics
## below
```

The `fmi.factor()` method calls `cmatrix()` internally, so explicitly invoking `fmi.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Fowlkes Mallows Index
fmi(...)

## Generic S3 method
## for weighted Fowlkes Mallows Index
weighted.fmi(...)
```

Arguments

```
...      Arguments passed on to fmi.factor, weighted.fmi.factor, fmi.cmatrix
        actual, predicted A pair of <integer> or <factor> vectors of length  $n$ , and  $k$ 
        levels.
w A <double> vector of sample weights.
x A confusion matrix created cmatrix().
```

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

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

```

classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::fmi(
  actual = actual_classes,
  predicted = predicted_classes
)

```

fmi.cmatrix

Fowlkes Mallows Index

Description

A generic S3 function to compute the *fowlkes mallows index* score for a classification model. This function dispatches to S3 methods in `fmi()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fmi()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fmi()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_fmi <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fmi(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate fowlkes mallows index
## via S3 dispatching
fmi(confusion_matrix)

## additional performance metrics
## below
```

The `fmi.factor()` method calls `cmatrix()` internally, so explicitly invoking `fmi.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

<code>x</code>	A confusion matrix created <code>cmatrix()</code> .
<code>...</code>	Arguments passed into other methods.

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneerror\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::fmi(confusion_matrix)
```

fmi.factor

Fowlkes Mallows Index

Description

A generic S3 function to compute the *fowlkes mallows index* score for a classification model. This function dispatches to S3 methods in [fmi\(\)](#) and performs no input validation. If you supply **NA** values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fmi()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fmi()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fmi <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fmi(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate fowlkes mallows index
## via S3 dispatching
fmi(confusion_matrix)

## additional performance metrics
## below
```

The `fmi.factor()` method calls `cmatrix()` internally, so explicitly invoking `fmi.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

```
actual, predicted      A pair of <integer> or <factor> vectors of length n, and k levels.
...                   Arguments passed into other methods.
```

Value

A <double>-value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::fmi(
  actual = actual_classes,
  predicted = predicted_classes
)
```

Description

A generic S3 function to compute the *false positive rate* score for a classification model. This function dispatches to S3 methods in `fpr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fpr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fpr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fpr <- function(x, y, ...) {  
  stopifnot(  
    !anyNA(x), !anyNA(y),  
    length(x) == length(y)  
  )  
  fpr(x, y, ...)  
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix  
confusion_matrix <- cmatrix(actual, predicted)  
  
## evaluate false positive rate  
## via S3 dispatching  
fpr(confusion_matrix)  
  
## additional performance metrics  
## below
```

The `fpr.factor()` method calls `cmatrix()` internally, so explicitly invoking `fpr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for False Positive Rate
fpr(...)

## Generic S3 method
## for weighted False Positive Rate
weighted.fpr(...)
```

Arguments

... Arguments passed on to `fpr.factor`, `weighted.fpr.factor`, `fpr.cmatrix`

actual, predicted A pair of `<integer>` or `<factor>` vectors of length n , and k levels.

estimator An `<integer>`-value of length 1 (default: 0).

- 0 - a named `<double>`-vector of length k (class-wise)
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

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))`.

w A `<double>` vector of sample weights.

x A confusion matrix created `cmatrix()`.

Value

If estimator is given as

- 0 - a named `<double>` vector of length k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The false positive rate has other names depending on research field:

- Fallout, `fallout()`

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::fpr(
  actual = actual_classes,
  predicted = predicted_classes
)
```

Description

A generic S3 function to compute the *false positive rate* score for a classification model. This function dispatches to S3 methods in `fpr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fpr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fpr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fpr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fpr(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate false positive rate
## via S3 dispatching
fpr(confusion_matrix)

## additional performance metrics
## below
```

The `fpr.factor()` method calls `cmatrix()` internally, so explicitly invoking `fpr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
fpr(x, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

x	A confusion matrix created <code>cmatrix()</code> .
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The false positive rate has other names depending on research field:

- Fallout, `fallout()`

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `gmse()`, `hammingloss()`, `huberloss()`,

```
jaccard(), logloss(), maape(), mae(), mape(), mcc(), mpe(), mse(), nlr(), npv(), pinball(),
plr(), pr.curve(), precision(), rae(), recall(), relative.entropy(), rmse(), rmsle(),
roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(), zeroone loss()
```

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::fpr(confusion_matrix)
```

fpr.factor

False Positive Rate

Description

A generic S3 function to compute the *false positive rate* score for a classification model. This function dispatches to S3 methods in `fpr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fpr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fpr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fpr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fpr(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate false positive rate
## via S3 dispatching
fpr(confusion_matrix)

## additional performance metrics
## below
```

The `fpr.factor()` method calls `cmatrix()` internally, so explicitly invoking `fpr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
fpr(actual, predicted, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

<code>actual, predicted</code>	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
<code>estimator</code>	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
<code>na.rm</code>	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA))</code> ,

```

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)).
... Arguments passed into other methods.

```

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The false positive rate has other names depending on research field:

- Fallout, `fallout()`

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```

## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual

```

```
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::fpr(
  actual = actual_classes,
  predicted = predicted_classes
)
```

gmse

Geometric Mean Squared Error

Description

A generic S3 function to compute the *geometric mean squared error* score for a regression model. This function dispatches to S3 methods in `gmse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `gmse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `gmse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_gmse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  gmse(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Geometric Mean Squared Error
gmse(...)

## Generic S3 method
## for weighted Geometric Mean Squared Error
weighted.gmse(...)
```

Arguments

```
...      Arguments passed on to gmse.numeric, weighted.gmse.numeric
actual,predicted A pair of <double> vectors of length n.
w A <double> vector of sample weights.
```

Value

```
A <double> value
```

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
```

```
SLmetrics::gmse(
  actual    = actual_values,
  predicted = predicted_values
)
```

gmse.numeric

Geometric Mean Squared Error

Description

A generic S3 function to compute the *geometric mean squared error* score for a regression model. This function dispatches to S3 methods in `gmse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `gmse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `gmse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_gmse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  gmse(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

```
actual, predicted      A pair of <double> vectors of length n.
...                    Arguments passed into other methods
```

Value

A <double> value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::gmse(
  actual    = actual_values,
  predicted = predicted_values
)
```

hammingloss

Hamming Loss

Description

A generic S3 function to compute the *hamming loss* score for a classification model. This function dispatches to S3 methods in `hammingloss()` and performs no input validation. If you supply **NA** values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `hammingloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `hammingloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_hammingloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  hammingloss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate hamming loss
## via S3 dispatching
hammingloss(confusion_matrix)

## additional performance metrics
## below
```

The `hammingloss.factor()` method calls `cmatrix()` internally, so explicitly invoking `hammingloss.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Hamming Loss
hammingloss(...)

## Generic S3 method
## for weighted Hamming Loss
weighted.hammingloss(...)
```

Arguments

```
...           Arguments passed on to hammingloss.factor, weighted.hammingloss.factor,
              hammingloss.cmatrix
```

actual, predicted A pair of [integer](#) or [factor](#) vectors of [length](#) n , and k levels.

w A [double](#) vector of sample weights.

x A confusion matrix created [cmatrix\(\)](#).

Value

A [double](#)-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)
```

```
## Evaluate performance
SLmetrics::hammingloss(
  actual    = actual_classes,
  predicted = predicted_classes
)
```

hammingloss.cmatrix *Hamming Loss*

Description

A generic S3 function to compute the *hamming loss* score for a classification model. This function dispatches to S3 methods in `hammingloss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `hammingloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `hammingloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_hammingloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  hammingloss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate hamming loss
## via S3 dispatching
hammingloss(confusion_matrix)

## additional performance metrics
## below
```

The `hammingloss.factor()` method calls `cmatrix()` internally, so explicitly invoking `hammingloss.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

`x` A confusion matrix created `cmatrix()`.
`...` Arguments passed into other methods.

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
```

```

actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual    = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::hammingloss(confusion_matrix)

```

hammingloss.factor	<i>Hamming Loss</i>
--------------------	---------------------

Description

A generic S3 function to compute the *hamming loss* score for a classification model. This function dispatches to S3 methods in `hammingloss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `hammingloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `hammingloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_hammingloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  hammingloss(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate hamming loss
## via S3 dispatching
hammingloss(confusion_matrix)

## additional performance metrics
## below
```

The `hammingloss.factor()` method calls `cmatrix()` internally, so explicitly invoking `hammingloss.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

```
actual, predicted      A pair of <integer> or <factor> vectors of length  $n$ , and  $k$  levels.
...                   Arguments passed into other methods.
```

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone.loss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::hammingloss(
  actual = actual_classes,
  predicted = predicted_classes
)
```

huberloss

Huber Loss

Description

A generic S3 function to compute the *huber loss* score for a regression model. This function dispatches to S3 methods in [huberloss\(\)](#) and performs no input validation. If you supply [NA](#) values or vectors of unequal [length](#) (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because [huberloss\(\)](#) operates on raw pointers, pointer-level faults (e.g. from [NA](#) or mismatched [length](#)) occur before any R-level error handling. Wrapping calls in [try\(\)](#) or [tryCatch\(\)](#) will *not* prevent R-session crashes.

To guard against this, wrap `huberloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_huberloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  huberloss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Huber Loss
huberloss(...)

## Generic S3 method
## for weighted Huber Loss
weighted.huberloss(...)
```

Arguments

`...` Arguments passed on to `huberloss.numeric`, `weighted.huberloss.numeric`

`actual, predicted` A pair of `<double>` vectors of `length` n .

`delta` A `<double>`-vector of `length` 1 (default: 1). The threshold value for switch between functions (see calculation).

`w` A `<double>` vector of sample weights.

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::huberloss(
  actual = actual_values,
  predicted = predicted_values
)
```

huberloss.numeric	<i>Huber Loss</i>
-------------------	-------------------

Description

A generic S3 function to compute the *huber loss* score for a regression model. This function dispatches to S3 methods in `huberloss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `huberloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `huberloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_huberloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  huberloss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted	A pair of <code><double></code> vectors of <code>length</code> n .
delta	A <code><double></code> -vector of <code>length</code> 1 (default: 1). The threshold value for switch between functions (see calculation).
...	Arguments passed into other methods

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
```

```

predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::huberloss(
  actual    = actual_values,
  predicted = predicted_values
)

```

jaccard

Jaccard Index

Description

A generic S3 function to compute the *jaccard index* score for a classification model. This function dispatches to S3 methods in `jaccard()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `jaccard()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `jaccard()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_jaccard <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  jaccard(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```

## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate jaccard index
## via S3 dispatching
jaccard(confusion_matrix)

```

```
## additional performance metrics
## below
```

The `jaccard.factor()` method calls `cmatrix()` internally, so explicitly invoking `jaccard.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Jaccard Index
jaccard(...)

## Generic S3 method
## for weighted Jaccard Index
weighted.jaccard(...)
```

Arguments

... Arguments passed on to `jaccard.factor`, `weighted.jaccard.factor`, `jaccard.cmatrix`

actual, predicted A pair of `<integer>` or `<factor>` vectors of `length` n , and k levels.

estimator An `<integer>`-value of `length` 1 (default: 0).

- 0 - a named `<double>`-vector of `length` k (class-wise)
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

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))`.

w A `<double>` vector of sample weights.

x A confusion matrix created `cmatrix()`.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The specificity has other names depending on research field:

- Critical Success Index, `csi()`
- Threat Score, `tscore()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::jaccard(
  actual = actual_classes,
  predicted = predicted_classes
)
```

jaccard.cmatrix

*Jaccard Index***Description**

A generic S3 function to compute the *jaccard index* score for a classification model. This function dispatches to S3 methods in `jaccard()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `jaccard()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `jaccard()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_jaccard <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  jaccard(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate jaccard index
## via S3 dispatching
jaccard(confusion_matrix)

## additional performance metrics
## below
```

The `jaccard.factor()` method calls `cmatrix()` internally, so explicitly invoking `jaccard.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
jaccard(x, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

x	A confusion matrix created <code>cmatrix()</code> .
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The specificity has other names depending on research field:

- Critical Success Index, `csi()`
- Threat Score, `tscore()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::jaccard(confusion_matrix)
```

Description

A generic S3 function to compute the *jaccard index* score for a classification model. This function dispatches to S3 methods in `jaccard()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `jaccard()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `jaccard()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_jaccard <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  jaccard(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate jaccard index
## via S3 dispatching
jaccard(confusion_matrix)

## additional performance metrics
## below
```

The `jaccard.factor()` method calls `cmatrix()` internally, so explicitly invoking `jaccard.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
jaccard(actual, predicted, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The specificity has other names depending on research field:

- Critical Success Index, `csi()`
- Threat Score, `tscore()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`,

```
deviance.tweedie(), dor(), fbeta(), fdr(), fer(), fmi(), fpr(), gmse(), hammingloss(),
huberloss(), logloss(), maape(), mae(), mape(), mcc(), mpe(), mse(), nlr(), npv(), pinball(),
plr(), pr.curve(), precision(), rae(), recall(), relative.entropy(), rmse(), rmsle(),
roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(), zeroone loss()
```

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::jaccard(
  actual = actual_classes,
  predicted = predicted_classes
)
```

logloss

Logarithmic Loss

Description

A generic S3 function to compute the *logarithmic loss* score for a classification model. This function dispatches to S3 methods in `logloss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `logloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `logloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_logloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  logloss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Logarithmic Loss
logloss(...)

## Generic S3 method
## for weighted Logarithmic Loss
weighted.logloss(...)
```

Arguments

... Arguments passed on to `logloss.integer`, `logloss.factor`, `weighted.logloss.integer`, `weighted.logloss.factor`

actual A vector `length` n , and k levels. Can be of `integer` or `factor`.

response A $n \times k$ `<double>`-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.

w A `<double>` vector of sample weights.

Value

A `<double>`

References

- MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.
- Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Other Entropy: `cross.entropy()`, `relative.entropy()`, `shannon.entropy()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted response
## probabilities
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

response <- runif(n = 1e3)

## Logloss
SLmetrics::logloss(
  actual = actual_classes,
  response = cbind(
    response,
    1 - response
  )
)

## Generate observed
## frequencies
actual_frequency <- sample(10L:100L, size = 1e3, replace = TRUE)

## Poisson Logloss
SLmetrics::logloss(
  actual = actual_frequency,
  response = response
)
```

logloss.factor

Logarithmic Loss

Description

A generic S3 function to compute the *logarithmic loss* score for a classification model. This function dispatches to S3 methods in `logloss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `logloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `logloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_logloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  logloss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

<code>actual</code>	A vector <code>length</code> n , and k levels. Can be of <code>integer</code> or <code>factor</code> .
<code>response</code>	A $n \times k$ <code><double></code> -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 <code>actual</code> , the second column to the second factor level, and so on.
<code>normalize</code>	A <code><logical></code> -value (default: <code>TRUE</code>). If <code>TRUE</code> , the mean cross-entropy across all observations is returned; otherwise, the sum of cross-entropies is returned.
<code>...</code>	Arguments passed into other methods.

Value

A <double>

References

MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.

Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Entropy: [cross.entropy\(\)](#), [relative.entropy\(\)](#), [shannon.entropy\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted response
## probabilities
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

response <- runif(n = 1e3)

## Evaluate performance
SLmetrics::logloss(
  actual = actual_classes,
  response = cbind(
    response,
    1 - response
  )
)
```

```

    )
  )

  ## Generate observed
  ## frequencies
  actual_frequency <- sample(10L:100L, size = 1e3, replace = TRUE)

  SLmetrics::logloss(
    actual    = actual_frequency,
    response  = response
  )

```

logloss.integer

Logarithmic Loss

Description

A generic S3 function to compute the *logarithmic loss* score for a classification model. This function dispatches to S3 methods in `logloss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `logloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `logloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_logloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  logloss(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

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

```

Arguments

actual	A vector length n , and k levels. Can be of integer or factor .
response	A $n \times k$ double -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.

Value

A [double](#)

References

MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.

Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Entropy: [cross.entropy\(\)](#), [relative.entropy\(\)](#), [shannon.entropy\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted response
## probabilities
```

```

actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

response <- runif(n = 1e3)

## Evaluate performance
SLmetrics::logloss(
  actual = actual_classes,
  response = cbind(
    response,
    1 - response
  )
)

## Generate observed
## frequencies
actual_frequency <- sample(10L:100L, size = 1e3, replace = TRUE)

SLmetrics::logloss(
  actual = actual_frequency,
  response = response
)

```

maape

Mean Arctangent Absolute Percentage Error

Description

A generic S3 function to compute the *mean arctangent absolute percentage error* score for a regression model. This function dispatches to S3 methods in `maape()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `maape()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `maape()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_maape <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
}

```

```

    )
    maape(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## Generic S3 method
## for Mean Arctangent Absolute Percentage Error
maape(...)

## Generic S3 method
## for weighted Mean Arctangent Absolute Percentage Error
weighted.maape(...)

```

Arguments

... Arguments passed on to `maape.numeric`, `weighted.maape.numeric`
 actual, predicted A pair of `<double>` vectors of length `n`.
 w A `<double>` vector of sample weights.

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::maape(
  actual = actual_values,
  predicted = predicted_values
)
```

maape.numeric

Mean Arctangent Absolute Percentage Error

Description

A generic S3 function to compute the *mean arctangent absolute percentage error* score for a regression model. This function dispatches to S3 methods in `maape()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `maape()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `maape()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_maape <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  maape(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted
 A pair of <double> vectors of length *n*.
 ... Arguments passed into other methods

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::maape(
  actual    = actual_values,
  predicted = predicted_values
)
```

mae

*Mean Absolute Error***Description**

A generic S3 function to compute the *mean absolute error* score for a regression model. This function dispatches to S3 methods in `mae()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mae()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mae()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_mae <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mae(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Mean Absolute Error
mae(...)

## Generic S3 method
## for weighted Mean Absolute Error
weighted.mae(...)
```

Arguments

... Arguments passed on to `mae.numeric`, `weighted.mae.numeric`
 actual, predicted A pair of `<double>` vectors of `length` *n*.
 w A `<double>` vector of sample weights.

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneerror\(\)](#)

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::mae(
  actual = actual_values,
  predicted = predicted_values
)
```

mae.numeric

Mean Absolute Error

Description

A generic S3 function to compute the *mean absolute error* score for a regression model. This function dispatches to S3 methods in [mae\(\)](#) and performs no input validation. If you supply **NA** values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mae()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mae()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_mae <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mae(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

```
actual, predicted      A pair of <double> vectors of length n.
...                   Arguments passed into other methods
```

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::mae(
  actual = actual_values,
  predicted = predicted_values
)
```

mape	<i>Mean Absolute Percentage Error</i>
------	---------------------------------------

Description

A generic S3 function to compute the *mean absolute percentage error* score for a regression model. This function dispatches to S3 methods in [mape\(\)](#) and performs no input validation. If you supply [NA](#) values or vectors of unequal [length](#) (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because [mape\(\)](#) operates on raw pointers, pointer-level faults (e.g. from [NA](#) or mismatched [length](#)) occur before any R-level error handling. Wrapping calls in [try\(\)](#) or [tryCatch\(\)](#) will *not* prevent R-session crashes.

To guard against this, wrap [mape\(\)](#) in a "safe" validator that checks for [NA](#) values and matching [length](#), for example:

```
safe_mape <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mape(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Mean Absolute Percentage Error
mape(...)

## Generic S3 method
## for weighted Mean Absolute Percentage Error
weighted.mape(...)
```

Arguments

```
...      Arguments passed on to mape.numeric, weighted.mape.numeric
actual,predicted A pair of <double> vectors of length n.
w A <double> vector of sample weights.
```

Value

```
A <double> value
```

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)
```

```
## Evaluate performance
SLmetrics::mape(
  actual    = actual_values,
  predicted = predicted_values
)
```

mape.numeric

Mean Absolute Percentage Error

Description

A generic S3 function to compute the *mean absolute percentage error* score for a regression model. This function dispatches to S3 methods in `mape()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mape()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mape()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_mape <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mape(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

`actual, predicted` A pair of `<double>` vectors of `length` *n*.

`...` Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::mape(
  actual = actual_values,
  predicted = predicted_values
)
```

mcc

Matthews Correlation Coefficient

Description

A generic S3 function to compute the *matthews correlation coefficient* score for a classification model. This function dispatches to S3 methods in `mcc()` and performs no input validation. If you supply NA values or vectors of unequal length (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mcc()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mcc()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_mcc <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mcc(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate matthews correlation coefficient
## via S3 dispatching
mcc(confusion_matrix)

## additional performance metrics
## below
```

The `mcc.factor()` method calls `cmatrix()` internally, so explicitly invoking `mcc.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Matthews Correlation Coefficient
mcc(...)

## Generic S3 method
## for weighted Matthews Correlation Coefficient
weighted.mcc(...)
```

Arguments

... Arguments passed on to `mcc.factor`, `weighted.mcc.factor`, `mcc.cmatrix`

actual, predicted A pair of `<integer>` or `<factor>` vectors of length n , and k levels.

w A `<double>` vector of sample weights.

x A confusion matrix created `cmatrix()`.

Value

A `<double>`-value

Other names

The Matthews Correlation Coefficient has other names depending on research field:

- ϕ -coefficient, `phi()`

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
```

```

    levels = c("Kebab", "Falafel")
  )

  predicted_classes <- factor(
    x = sample(x = classes, size = 1e3, replace = TRUE),
    levels = c("Kebab", "Falafel")
  )

  ## Evaluate performance
  SLmetrics::mcc(
    actual    = actual_classes,
    predicted = predicted_classes
  )

```

mcc.cmatrix

Matthews Correlation Coefficient

Description

A generic S3 function to compute the *matthews correlation coefficient* score for a classification model. This function dispatches to S3 methods in `mcc()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mcc()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mcc()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_mcc <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mcc(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate matthews correlation coefficient
## via S3 dispatching
mcc(confusion_matrix)
```

```
## additional performance metrics
## below
```

The `mcc.factor()` method calls `cmatrix()` internally, so explicitly invoking `mcc.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

<code>x</code>	A confusion matrix created <code>cmatrix()</code> .
<code>...</code>	Arguments passed into other methods.

Value

A `<double>`-value

Other names

The Matthews Correlation Coefficient has other names depending on research field:

- ϕ -coefficient, `phi()`

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::mcc(confusion_matrix)
```

mcc.factor

Matthews Correlation Coefficient

Description

A generic S3 function to compute the *matthews correlation coefficient* score for a classification model. This function dispatches to S3 methods in [mcc\(\)](#) and performs no input validation. If you supply [NA](#) values or vectors of unequal [length](#) (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mcc()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mcc()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_mcc <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mcc(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate matthews correlation coefficient
## via S3 dispatching
mcc(confusion_matrix)

## additional performance metrics
## below
```

The `mcc.factor()` method calls `cmatrix()` internally, so explicitly invoking `mcc.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

```
actual, predicted      A pair of <integer> or <factor> vectors of length n, and k levels.
...                   Arguments passed into other methods.
```

Value

A <double>-value

Other names

The Matthews Correlation Coefficient has other names depending on research field:

- ϕ -coefficient, [phi\(\)](#)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::mcc(
  actual    = actual_classes,
```

```

    predicted = predicted_classes
  )

```

mpe

Mean Percentage Error

Description

A generic S3 function to compute the *mean percentage error* score for a regression model. This function dispatches to S3 methods in `mpe()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mpe()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mpe()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_mpe <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mpe(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## Generic S3 method
## for Mean Percentage Error
mpe(...)

## Generic S3 method
## for weighted Mean Percentage Error
weighted.mpe(...)

```

Arguments

```

...      Arguments passed on to mpe.numeric, weighted.mpe.numeric
actual,predicted A pair of <double> vectors of length n.
w        A <double> vector of sample weights.

```

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::mpe(
  actual    = actual_values,
  predicted = predicted_values
)
```

Description

A generic S3 function to compute the *mean percentage error* score for a regression model. This function dispatches to S3 methods in `mpe()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mpe()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mpe()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_mpe <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mpe(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length n</code> .
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::mpe(
  actual = actual_values,
  predicted = predicted_values
)
```

mse

Mean Squared Error

Description

A generic S3 function to compute the *mean squared error* score for a regression model. This function dispatches to S3 methods in `mse()` and performs no input validation. If you supply **NA** values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mse()` operates on raw pointers, pointer-level faults (e.g. from **NA** or mismatched **length**) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mse()` in a "safe" validator that checks for **NA** values and matching **length**, for example:

```
safe_mse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
}
```

```

    )
    mse(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## Generic S3 method
## for Mean Squared Error
mse(...)

## Generic S3 method
## for weighted Mean Squared Error
weighted.mse(...)

```

Arguments

... Arguments passed on to `mse.numeric`, `weighted.mse.numeric`
 actual, predicted A pair of `<double>` vectors of `length` *n*.
 w A `<double>` vector of sample weights.

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::mse(
  actual = actual_values,
  predicted = predicted_values
)
```

mse.numeric

Mean Squared Error

Description

A generic S3 function to compute the *mean squared error* score for a regression model. This function dispatches to S3 methods in `mse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_mse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mse(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted
 A pair of <double> vectors of length *n*.
 ... Arguments passed into other methods

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneerror()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::mse(
  actual    = actual_values,
  predicted = predicted_values
)
```

Description

A generic S3 function to compute the *negative likelihood ratio* score for a classification model. This function dispatches to S3 methods in `nlr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `nlr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `nlr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_nlr <- function(x, y, ...) {  
  stopifnot(  
    !anyNA(x), !anyNA(y),  
    length(x) == length(y)  
  )  
  nlr(x, y, ...)  
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix  
confusion_matrix <- cmatrix(actual, predicted)  
  
## evaluate negative likelihood ratio  
## via S3 dispatching  
nlr(confusion_matrix)  
  
## additional performance metrics  
## below
```

The `nlr.factor()` method calls `cmatrix()` internally, so explicitly invoking `nlr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Negative Likelihood Ratio
nlr(...)

## Generic S3 method
## for weighted Negative Likelihood Ratio
weighted.nlr(...)
```

Arguments

... Arguments passed on to `nlr.factor`, `weighted.nlr.factor`, `nlr.cmatrix`
 actual, predicted A pair of <integer> or <factor> vectors of length n , and k
 levels.
 w A <double> vector of sample weights.
 x A confusion matrix created `cmatrix()`.

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

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

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::nlr(
  actual = actual_classes,
  predicted = predicted_classes
)
```

nlr.cmatrix

Negative Likelihood Ratio

Description

A generic S3 function to compute the *negative likelihood ratio* score for a classification model. This function dispatches to S3 methods in `nlr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `nlr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `nlr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_nlr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  nlr(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate negative likelihood ratio
## via S3 dispatching
nlr(confusion_matrix)

## additional performance metrics
## below
```

The `nlr.factor()` method calls `cmatrix()` internally, so explicitly invoking `nlr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

<code>x</code>	A confusion matrix created <code>cmatrix()</code> .
<code>...</code>	Arguments passed into other methods.

Value

A `<double>`-value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

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

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::nlr(confusion_matrix)
```

Description

A generic S3 function to compute the *negative likelihood ratio* score for a classification model. This function dispatches to S3 methods in `nlr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `nlr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `nlr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_nlr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  nlr(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate negative likelihood ratio
## via S3 dispatching
nlr(confusion_matrix)

## additional performance metrics
## below
```

The `nlr.factor()` method calls `cmatrix()` internally, so explicitly invoking `nlr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

```
actual, predicted      A pair of <integer> or <factor> vectors of length  $n$ , and  $k$  levels.
...                   Arguments passed into other methods.
```

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

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

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
```

```

    x = sample(x = classes, size = 1e3, replace = TRUE),
    levels = c("Kebab", "Falafel")
  )

  predicted_classes <- factor(
    x = sample(x = classes, size = 1e3, replace = TRUE),
    levels = c("Kebab", "Falafel")
  )

  ## Evaluate performance
  SLmetrics::nlr(
    actual    = actual_classes,
    predicted = predicted_classes
  )

```

npv

*Negative Predictive Value***Description**

A generic S3 function to compute the *negative predictive value* score for a classification model. This function dispatches to S3 methods in `npv()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `npv()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `npv()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_npv <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  npv(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate negative predictive value
## via S3 dispatching
npv(confusion_matrix)

## additional performance metrics
## below
```

The `npv.factor()` method calls `cmatrix()` internally, so explicitly invoking `npv.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Negative Predictive Value
npv(...)

## Generic S3 method
## for weighted Negative Predictive Value
weighted.npv(...)
```

Arguments

... Arguments passed on to `npv.factor`, `weighted.npv.factor`, `npv.cmatrix`

actual, predicted A pair of `<integer>` or `<factor>` vectors of length n , and k levels.

estimator An `<integer>`-value of length 1 (default: 0).

- 0 - a named `<double>`-vector of length k (class-wise)
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

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))`.

w A `<double>` vector of sample weights.

x A confusion matrix created `cmatrix()`.

Value

If estimator is given as

- 0 - a named `<double>` vector of length k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::npv(
  actual = actual_classes,
  predicted = predicted_classes
)
```

Description

A generic S3 function to compute the *negative predictive value* score for a classification model. This function dispatches to S3 methods in `npv()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `npv()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `npv()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_npv <- function(x, y, ...) {  
  stopifnot(  
    !anyNA(x), !anyNA(y),  
    length(x) == length(y)  
  )  
  npv(x, y, ...)  
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix  
confusion_matrix <- cmatrix(actual, predicted)  
  
## evaluate negative predictive value  
## via S3 dispatching  
npv(confusion_matrix)  
  
## additional performance metrics  
## below
```

The `npv.factor()` method calls `cmatrix()` internally, so explicitly invoking `npv.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
npv(x, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

x	A confusion matrix created <code>cmatrix()</code> .
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`,

```
pinball(),plr(),pr.curve(),precision(),rae(),recall(),relative.entropy(),rmse(),
rmsle(),roc.curve(),rrmse(),rrse(),rsq(),shannon.entropy(),smape(),specificity(),
zerooneerror()
```

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual    = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::npv(confusion_matrix)
```

npv.factor

Negative Predictive Value

Description

A generic S3 function to compute the *negative predictive value* score for a classification model. This function dispatches to S3 methods in `npv()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `npv()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `npv()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_npv <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  npv(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate negative predictive value
## via S3 dispatching
npv(confusion_matrix)

## additional performance metrics
## below
```

The `npv.factor()` method calls `cmatrix()` internally, so explicitly invoking `npv.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
npv(actual, predicted, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

<code>actual, predicted</code>	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
<code>estimator</code>	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
<code>na.rm</code>	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA))</code> ,

```

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)).
... Arguments passed into other methods.

```

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```

## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

```

```

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::npv(
  actual = actual_classes,
  predicted = predicted_classes
)

```

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.

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).

fevc 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).

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.

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).

References

Palechor, Fabio Mendoza, and Alexis De la Hoz Manotas. "Dataset for estimation of obesity levels based on eating habits and physical condition in individuals from Colombia, Peru and Mexico." Data in brief 25 (2019): 104344.

OpenMP

Control OpenMP

Description

Enable or disable OpenMP parallelization for computations.

Disclaimer:

This toggle is a brute-force implementation and does **not** guard against data races or nested parallel regions. Nested OpenMP regions can introduce subtle race conditions if multiple layers of parallelism access shared data concurrently. If you combine this package's OpenMP switch with other parallel machine-learning routines, you may encounter undefined behavior.

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](#).

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

Pinball Loss

Description

A generic S3 function to compute the *pinball loss* score for a regression model. This function dispatches to S3 methods in [pinball\(\)](#) and performs no input validation. If you supply [NA](#) values or vectors of unequal [length](#) (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because [pinball\(\)](#) operates on raw pointers, pointer-level faults (e.g. from [NA](#) or mismatched [length](#)) occur before any R-level error handling. Wrapping calls in [try\(\)](#) or [tryCatch\(\)](#) will *not* prevent R-session crashes.

To guard against this, wrap [pinball\(\)](#) in a "safe" validator that checks for [NA](#) values and matching [length](#), for example:

```
safe_pinball <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
```

```

    length(x) == length(y)
  )
  pinball(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## Generic S3 method
## for Pinball Loss
pinball(...)

## Generic S3 method
## for weighted Pinball Loss
weighted.pinball(...)

```

Arguments

... Arguments passed on to `pinball.numeric`, `weighted.pinball.numeric`

actual, predicted A pair of `<double>` vectors of `length` n .

alpha A `<double>`-value of `length` 1 (default: 0.5). The slope of the pinball loss function.

deviance A `<logical>`-value of `length` 1 (default: `FALSE`). If `TRUE` the function returns the D^2 loss.

w A `<double>` vector of sample weights.

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`,

```
deviance.tweedie(), dor(), fbeta(), fdr(), fer(), fmi(), fpr(), gmse(), hammingloss(),
huberloss(), jaccard(), logloss(), maape(), mae(), mape(), mcc(), mpe(), mse(), nlr(),
npv(), plr(), pr.curve(), precision(), rae(), recall(), relative.entropy(), rmse(), rmsle(),
roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(), zeroone loss()
```

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::pinball(
  actual = actual_values,
  predicted = predicted_values
)
```

pinball.numeric	<i>Pinball Loss</i>
-----------------	---------------------

Description

A generic S3 function to compute the *pinball loss* score for a regression model. This function dispatches to S3 methods in `pinball()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `pinball()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `pinball()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_pinball <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  pinball(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted	A pair of <double> vectors of length n .
alpha	A <double>-value of length 1 (default: 0.5). The slope of the pinball loss function.
deviance	A <logical>-value of length 1 (default: FALSE). If TRUE the function returns the D^2 loss.
...	Arguments passed into other methods

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)
```

```
## Evaluate performance
SLmetrics::pinball(
  actual    = actual_values,
  predicted = predicted_values
)
```

plr

Positive Likelihood Ratio

Description

A generic S3 function to compute the *positive likelihood ratio* score for a classification model. This function dispatches to S3 methods in `plr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `plr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `plr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_plr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  plr(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate positive likelihood ratio
## via S3 dispatching
plr(confusion_matrix)

## additional performance metrics
## below
```

The `plr.factor()` method calls `cmatrix()` internally, so explicitly invoking `plr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Positive Likelihood Ratio
plr(...)

## Generic S3 method
## for weighted Positive Likelihood Ratio
weighted.plr(...)

## Generic S3 method
## for weighted Diagnostic Odds Ratio
weighted.plr(...)
```

Arguments

... Arguments passed on to `plr.factor`, `weighted.plr.factor`, `plr.cmatrix`
 actual, predicted A pair of `<integer>` or `<factor>` vectors of length n , and k
 levels.
 w A `<double>` vector of sample weights.
 x A confusion matrix created `cmatrix()`.

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

The `nlr()`-function for the Negative Likelihood Ratio (LR-)

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`,

```
deviance.tweedie(), dor(), fbeta(), fdr(), fer(), fmi(), fpr(), gmse(), hammingloss(),
huberloss(), jaccard(), logloss(), maape(), mae(), mape(), mcc(), mpe(), mse(), nlr(),
npv(), pinball(), pr.curve(), precision(), rae(), recall(), relative.entropy(), rmse(),
rmsle(), roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(),
zerooneerror()
```

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::plr(
  actual = actual_classes,
  predicted = predicted_classes
)
```

plr.cmatrix

Positive Likelihood Ratio

Description

A generic S3 function to compute the *positive likelihood ratio* score for a classification model. This function dispatches to S3 methods in `plr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `plr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `plr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_plr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  plr(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate positive likelihood ratio
## via S3 dispatching
plr(confusion_matrix)

## additional performance metrics
## below
```

The `plr.factor()` method calls `cmatrix()` internally, so explicitly invoking `plr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

<code>x</code>	A confusion matrix created <code>cmatrix()</code> .
<code>...</code>	Arguments passed into other methods.

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

The `nlr()`-function for the Negative Likelihood Ratio (LR-)

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual    = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::plr(confusion_matrix)
```

Description

A generic S3 function to compute the *positive likelihood ratio* score for a classification model. This function dispatches to S3 methods in `plr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `plr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `plr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_plr <- function(x, y, ...) {  
  stopifnot(  
    !anyNA(x), !anyNA(y),  
    length(x) == length(y)  
  )  
  plr(x, y, ...)  
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix  
confusion_matrix <- cmatrix(actual, predicted)  
  
## evaluate positive likelihood ratio  
## via S3 dispatching  
plr(confusion_matrix)  
  
## additional performance metrics  
## below
```

The `plr.factor()` method calls `cmatrix()` internally, so explicitly invoking `plr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

```
actual, predicted      A pair of <integer> or <factor> vectors of length  $n$ , and  $k$  levels.
...                   Arguments passed into other methods.
```

Value

```
A <double>-value
```

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

The `nlr()`-function for the Negative Likelihood Ratio (LR-)

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
```

```

    x = sample(x = classes, size = 1e3, replace = TRUE),
    levels = c("Kebab", "Falafel")
  )

  predicted_classes <- factor(
    x = sample(x = classes, size = 1e3, replace = TRUE),
    levels = c("Kebab", "Falafel")
  )

  ## Evaluate performance
  SLmetrics::plr(
    actual    = actual_classes,
    predicted = predicted_classes
  )

```

pr.curve

*Precision Recall Curve***Description**

A generic S3 function to compute the *precision recall curve* score for a classification model. This function dispatches to S3 methods in `pr.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `pr.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `pr.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_pr.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  pr.curve(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Area under the curve:

Use `auc.pr.curve` for calculating the area under the curve directly.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```
## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate precision recall curve
pr.curve(actual, response, indices = indices)
```

Usage

```
## Generic S3 method
## for Precision Recall Curve
pr.curve(...)

## Generic S3 method
## for weighted Precision Recall Curve
weighted.pr.curve(...)
```

Arguments

... Arguments passed on to `pr.curve.factor`, `weighted.pr.curve.factor`

actual A vector `length` n , and k levels. Can be of `integer` or `factor`.

response A $n \times k$ `<double>`-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.

indices An optional $n \times k$ matrix of `<integer>` values of sorted response probability indices.

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

w A `<double>` vector of sample weights.

Value

A `data.frame` on the following form,

threshold	<code><numeric></code> Thresholds used to determine <code>recall()</code> and <code>precision()</code>
level	<code><character></code> The level of the actual <code><factor></code>
label	<code><character></code> The levels of the actual <code><factor></code>
recall	<code><numeric></code> The recall
precision	<code><numeric></code> The precision

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
  rbeta(sum(actual_classes == "Kebab"), 2, 5),
  rbeta(sum(actual_classes == "Falafel"), 5, 2)
)

## Construct response
## matrix
probability_matrix <- cbind(
  response_probabilities,
  1 - response_probabilities
)
```

```

)

## Visualize precision recall curve

plot(
  SLmetrics::pr.curve(
    actual = actual_classes,
    response = probability_matrix
  )
)

```

pr.curve.factor

Precision Recall Curve

Description

A generic S3 function to compute the *precision recall curve* score for a classification model. This function dispatches to S3 methods in `pr.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `pr.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `pr.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_pr.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  pr.curve(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Area under the curve:

Use `auc.pr.curve` for calculating the area under the curve directly.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```
## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate precision recall curve
pr.curve(actual, response, indices = indices)
```

Usage

```
## S3 method for class 'factor'
pr.curve(actual, response, thresholds = NULL, indices = NULL, ...)
```

Arguments

actual	A vector length n , and k levels. Can be of integer or factor .
response	A $n \times k$ <double> -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.
thresholds	An optional <double> vector of length n (default: NULL).
indices	An optional $n \times k$ matrix of <integer> values of sorted response probability indices.
...	Arguments passed into other methods.

Value

A **data.frame** on the following form,

threshold	<numeric> Thresholds used to determine recall() and precision()
level	<character> The level of the actual <factor>
label	<character> The levels of the actual <factor>
recall	<numeric> The recall
precision	<numeric> The precision

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
  rbeta(sum(actual_classes == "Kebab"), 2, 5),
  rbeta(sum(actual_classes == "Falafel"), 5, 2)
)

## Construct response
## matrix
probability_matrix <- cbind(
  response_probabilities,
  1 - response_probabilities
)

## Visualize

plot(
  SLmetrics::pr.curve(
    actual = actual_classes,
    response = probability_matrix
```

```
)
)
```

```
precision
```

```
Precision
```

Description

A generic S3 function to compute the *precision* score for a classification model. This function dispatches to S3 methods in `precision()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `precision()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `precision()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_precision <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  precision(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate precision
## via S3 dispatching
precision(confusion_matrix)

## additional performance metrics
## below
```

The `precision.factor()` method calls `cmatrix()` internally, so explicitly invoking `precision.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Precision
precision(...)

## Generic S3 method
## for weighted Precision
weighted.precision(...)
```

Arguments

... Arguments passed on to `precision.factor`, `weighted.precision.factor`, `precision.cmatrix`

actual, predicted A pair of `<integer>` or `<factor>` vectors of `length` n , and k levels.

estimator An `<integer>`-value of `length` 1 (default: 0).

- 0 - a named `<double>`-vector of `length` k (class-wise)
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

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))`.

w A `<double>` vector of sample weights.

x A confusion matrix created `cmatrix()`.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The precision has other names depending on research field:

- Positive Predictive Value, `ppv()`

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::precision(
  actual    = actual_classes,
  predicted = predicted_classes
)
```

Description

A generic S3 function to compute the *precision* score for a classification model. This function dispatches to S3 methods in `precision()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `precision()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `precision()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_precision <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  precision(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate precision
## via S3 dispatching
precision(confusion_matrix)

## additional performance metrics
## below
```

The `precision.factor()` method calls `cmatrix()` internally, so explicitly invoking `precision.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
precision(x, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

x	A confusion matrix created <code>cmatrix()</code> .
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The precision has other names depending on research field:

- Positive Predictive Value, `ppv()`

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`,

```
huberloss(), jaccard(), logloss(), maape(), mae(), mape(), mcc(), mpe(), mse(), nlr(),
npv(), pinball(), plr(), pr.curve(), rae(), recall(), relative.entropy(), rmse(), rmsle(),
roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(), zerooneerror()
```

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::precision(confusion_matrix)
```

precision.factor	<i>Precision</i>
------------------	------------------

Description

A generic S3 function to compute the *precision* score for a classification model. This function dispatches to S3 methods in `precision()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `precision()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `precision()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_precision <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  precision(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate precision
## via S3 dispatching
precision(confusion_matrix)

## additional performance metrics
## below
```

The `precision.factor()` method calls `cmatrix()` internally, so explicitly invoking `precision.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
precision(actual, predicted, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

<code>actual, predicted</code>	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
<code>estimator</code>	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
<code>na.rm</code>	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA))</code> ,

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)).

... Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The precision has other names depending on research field:

- Positive Predictive Value, `ppv()`

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
```

```
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::precision(
  actual = actual_classes,
  predicted = predicted_classes
)
```

preorder

*Preorder Matrices***Description**

A generic S3 function for something long. This function dispatches to S3 methods in `preorder()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `preorder()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `preorder()` in a “safe” validator that checks for `NA` values and matching `length`, for example:

```
safe_preorder <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  preorder(x, y, ...)
}
```

Usage

```
## Generic S3 method
## for Preorder Matrices
preorder(...)
```

Arguments

... Arguments passed on to `preorder.matrix`
 x A `<matrix>` to be sorted
 decreasing A `<logical>`

Value

A container of sorted indices

See Also

Other Utilities: `presort()`

preorder.matrix	<i>Preorder Matrices</i>
-----------------	--------------------------

Description

A generic S3 function for something long. This function dispatches to S3 methods in `preorder()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `preorder()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `preorder()` in a “safe” validator that checks for `NA` values and matching `length`, for example:

```
safe_preorder <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  preorder(x, y, ...)
}
```

Usage

```
## S3 method for class 'matrix'
preorder(x, decreasing = FALSE, ...)
```

Arguments

x A `<matrix>` to be sorted
 decreasing A `<logical>`
 ... Arguments passed into other methods

Value

A [<matrix>](#) of same dimensions as the input [<matrix>](#)

See Also

Other Utilities: [presort\(\)](#)

presort

Presort Matrices

Description

A generic S3 function for something long. This function dispatches to S3 methods in [presort\(\)](#) and performs no input validation. If you supply [NA](#) values or vectors of unequal [length](#) (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because [presort\(\)](#) operates on raw pointers, pointer-level faults (e.g. from [NA](#) or mismatched [length](#)) occur before any R-level error handling. Wrapping calls in [try\(\)](#) or [tryCatch\(\)](#) will *not* prevent R-session crashes.

To guard against this, wrap [presort\(\)](#) in a “safe” validator that checks for [NA](#) values and matching [length](#), for example:

```
safe_presort <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  presort(x, y, ...)
}
```

Usage

```
## Generic S3 method
## for Presort Matrices
presort(...)
```

Arguments

... Arguments passed on to [presort.matrix](#)
 x A [<matrix>](#) to be sorted
 decreasing A [<logical>](#)

Value

A sorted container

See Also

Other Utilities: [preorder\(\)](#)

presort.matrix

Presort Matrices

Description

A generic S3 function for something long. This function dispatches to S3 methods in [presort\(\)](#) and performs no input validation. If you supply [NA](#) values or vectors of unequal [length](#) (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because [presort\(\)](#) operates on raw pointers, pointer-level faults (e.g. from [NA](#) or mismatched [length](#)) occur before any R-level error handling. Wrapping calls in [try\(\)](#) or [tryCatch\(\)](#) will *not* prevent R-session crashes.

To guard against this, wrap [presort\(\)](#) in a “safe” validator that checks for [NA](#) values and matching [length](#), for example:

```
safe_presort <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  presort(x, y, ...)
}
```

Usage

```
## S3 method for class 'matrix'
presort(x, decreasing = FALSE, ...)
```

Arguments

<code>x</code>	A matrix to be sorted
<code>decreasing</code>	A logical
<code>...</code>	Arguments passed into other methods

Value

A [matrix](#) of same dimensions as the input [matrix](#)

See Also

Other Utilities: [preorder\(\)](#)

rae

*Relative Absolute Error***Description**

A generic S3 function to compute the *relative absolute error* score for a regression model. This function dispatches to S3 methods in `rae()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rae()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rae()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rae <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rae(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Relative Absolute Error
rae(...)

## Generic S3 method
## for weighted Relative Absolute Error
weighted.rae(...)
```

Arguments

... Arguments passed on to `rae.numeric`, `weighted.rae.numeric`
 actual, predicted A pair of `<double>` vectors of `length` *n*.
 w A `<double>` vector of sample weights.

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneerror()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rae(
  actual = actual_values,
  predicted = predicted_values
)
```

rae.numeric

Relative Absolute Error

Description

A generic S3 function to compute the *relative absolute error* score for a regression model. This function dispatches to S3 methods in `rae()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rae()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rae()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rae <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rae(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

```
actual, predicted      A pair of <double> vectors of length n.
...                   Arguments passed into other methods
```

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rae(
  actual = actual_values,
  predicted = predicted_values
)
```

recall

Recall

Description

A generic S3 function to compute the *recall* score for a classification model. This function dispatches to S3 methods in `recall()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `recall()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `recall()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_recall <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  recall(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate recall
## via S3 dispatching
recall(confusion_matrix)

## additional performance metrics
## below
```

The `recall.factor()` method calls `cmatrix()` internally, so explicitly invoking `recall.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Recall
recall(...)

## Generic S3 method
## for weighted Recall
weighted.recall(...)
```

Arguments

```
...      Arguments passed on to recall.factor, weighted.recall.factor, recall.cmatrix
actual,predicted A pair of <integer> or <factor> vectors of length  $n$ , and  $k$ 
              levels.
estimator An <integer>-value of length 1 (default: 0).
      • 0 - a named <double>-vector of length  $k$  (class-wise)
      • 1 - a <double> value (Micro averaged metric)
      • 2 - a <double> value (Macro averaged metric)
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)).
w A <double> vector of sample weights.
x A confusion matrix created cmatrix().
```

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The Recall has other names depending on research field:

- Sensitivity, `sensitivity()`
- True Positive Rate, `tpr()`

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
```

```

    levels = c("Kebab", "Falafel")
  )

  predicted_classes <- factor(
    x = sample(x = classes, size = 1e3, replace = TRUE),
    levels = c("Kebab", "Falafel")
  )

  ## Evaluate performance
  SLmetrics::recall(
    actual    = actual_classes,
    predicted = predicted_classes
  )

```

recall.cmatrix

Recall

Description

A generic S3 function to compute the *recall* score for a classification model. This function dispatches to S3 methods in `recall()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `recall()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `recall()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_recall <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  recall(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate recall
## via S3 dispatching
recall(confusion_matrix)

## additional performance metrics
## below
```

The `recall.factor()` method calls `cmatrix()` internally, so explicitly invoking `recall.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
recall(x, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

<code>x</code>	A confusion matrix created <code>cmatrix()</code> .
<code>estimator</code>	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> <code>k</code> (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
<code>na.rm</code>	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
<code>...</code>	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` `k`
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The Recall has other names depending on research field:

- Sensitivity, `sensitivity()`
- True Positive Rate, `tpr()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual    = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
```

```
SLmetrics::recall(confusion_matrix)
```

```
recall.factor
```

```
Recall
```

Description

A generic S3 function to compute the *recall* score for a classification model. This function dispatches to S3 methods in `recall()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `recall()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `recall()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_recall <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  recall(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate recall
## via S3 dispatching
recall(confusion_matrix)

## additional performance metrics
## below
```

The `recall.factor()` method calls `cmatrix()` internally, so explicitly invoking `recall.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
recall(actual, predicted, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

`actual, predicted` A pair of `<integer>` or `<factor>` vectors of `length` n , and k levels.

`estimator` An `<integer>`-value of `length` 1 (default: 0).

- 0 - a named `<double>`-vector of `length` k (class-wise)
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

`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))`.

`...` Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The Recall has other names depending on research field:

- Sensitivity, `sensitivity()`
- True Positive Rate, `tpr()`

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::recall(
  actual    = actual_classes,
  predicted = predicted_classes
)
```

relative.entropy

Relative Entropy

Description

A generic S3 function to compute the *relative entropy* score for a classification model. This function dispatches to S3 methods in `relative.entropy()` and performs no input validation. If you supply

NA values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `relative.entropy()` operates on raw pointers, pointer-level faults (e.g. from **NA** or mismatched **length**) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `relative.entropy()` in a "safe" validator that checks for **NA** values and matching **length**, for example:

```
safe_relative.entropy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  relative.entropy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Relative Entropy
relative.entropy(...)
```

Arguments

`...` Arguments passed on to `relative.entropy.matrix`

`pk, qk` A pair of **<double>** matrices of **length** *n* of empirical probabilities *p* and estimated probabilities *q*.

`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).

`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.

Value

A **<double>** value or vector:

- A single **<double>** value (length 1) if `dim == 0`.
- A **<double>** vector with length equal to the **length** of columns if `dim == 1`.
- A **<double>** vector with length equal to the **length** of rows if `dim == 2`.

References

- MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.
- Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Entropy: [cross.entropy\(\)](#), [logloss\(\)](#), [shannon.entropy\(\)](#)

Examples

```
## generate valid probability
## distributions
rand.sum <- function(n) {
  x <- sort(runif( n-1 ))
  c(x,1) - c(0, x)
}

## empirical and
## predicted probabilities
set.seed(1903)
pk <- t(replicate(200,rand.sum(5)))
qk <- t(replicate(200,rand.sum(5)))

## entropy
relative.entropy(
  pk = pk,
  qk = qk
)
```

relative.entropy.matrix

Relative Entropy

Description

A generic S3 function to compute the *relative entropy* score for a classification model. This function dispatches to S3 methods in `relative.entropy()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `relative.entropy()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `relative.entropy()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_relative.entropy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  relative.entropy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## S3 method for class 'matrix'
relative.entropy(pk, qk, dim = 0L, normalize = FALSE, ...)
```

Arguments

pk, qk	A pair of <code><double></code> matrices of <code>length</code> n of empirical probabilities p and estimated probabilities q .
dim	An <code><integer></code> value of <code>length</code> 1 (Default: 0). Defines the dimension along which to calculate the entropy (0: total, 1: row-wise, 2: column-wise).
normalize	A <code><logical></code> -value (default: <code>TRUE</code>). If <code>TRUE</code> , the mean cross-entropy across all observations is returned; otherwise, the sum of cross-entropies is returned.
...	Arguments passed into other methods.

Value

A [<double>](#) value or vector:

- A single [<double>](#) value (length 1) if `dim == 0`.
- A [<double>](#) vector with length equal to the [length](#) of columns if `dim == 1`.
- A [<double>](#) vector with length equal to the [length](#) of rows if `dim == 2`.

References

MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.

Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Entropy: [cross.entropy\(\)](#), [logloss\(\)](#), [shannon.entropy\(\)](#)

Examples

```
## generate valid probability
## distributions
rand.sum <- function(n) {
  x <- sort(runif( n-1 ))
  c(x,1) - c(0, x)
}

## empirical and
## predicted probabilities
set.seed(1903)
pk <- t(replicate(200,rand.sum(5)))
qk <- t(replicate(200,rand.sum(5)))

## entropy
relative.entropy(
  pk = pk,
```

```

    qk = qk
  )

```

rmse

Root Mean Squared Error

Description

A generic S3 function to compute the *root mean squared error* score for a regression model. This function dispatches to S3 methods in `rmse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rmse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rmse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_rmse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rmse(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## Generic S3 method
## for Root Mean Squared Error
rmse(...)

## Generic S3 method
## for weighted Root Mean Squared Error
weighted.rmse(...)

```

Arguments

```

...      Arguments passed on to rmse.numeric, weighted.rmse.numeric
actual,predicted A pair of <double> vectors of length n.
w        A <double> vector of sample weights.

```

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rmse(
  actual    = actual_values,
  predicted = predicted_values
)
```

Description

A generic S3 function to compute the *root mean squared error* score for a regression model. This function dispatches to S3 methods in `rmse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rmse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rmse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rmse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rmse(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length n</code> .
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rmse(
  actual = actual_values,
  predicted = predicted_values
)
```

rmsle

*Root Mean Squared Logarithmic Error***Description**

A generic S3 function to compute the *root mean squared logarithmic error* score for a regression model. This function dispatches to S3 methods in `rmsle()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rmsle()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rmsle()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rmsle <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
```

```

    )
    rmsle(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## Generic S3 method
## for Root Mean Squared Logarithmic Error
rmsle(...)

## Generic S3 method
## for weighted Root Mean Squared Logarithmic Error
weighted.rmsle(...)

```

Arguments

... Arguments passed on to `rmsle.numeric`, `weighted.rmsle.numeric`
 actual, predicted A pair of `<double>` vectors of length `n`.
 w A `<double>` vector of sample weights.

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rmsle(
  actual = actual_values,
  predicted = predicted_values
)
```

rmsle.numeric

Root Mean Squared Logarithmic Error

Description

A generic S3 function to compute the *root mean squared logarithmic error* score for a regression model. This function dispatches to S3 methods in `rmsle()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rmsle()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rmsle()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rmsle <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rmsle(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted
 A pair of [<double>](#) vectors of [length](#) *n*.
 ... Arguments passed into other methods

Value

A [<double>](#) value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneerror\(\)](#)

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rmsle(
  actual    = actual_values,
  predicted = predicted_values
)
```

roc.curve

*Reciever Operator Characteristics***Description**

A generic S3 function to compute the *reciever operator characteristics* score for a classification model. This function dispatches to S3 methods in `roc.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `roc.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `roc.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_roc.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  roc.curve(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Area under the curve:

Use `auc.roc.curve` for calculating the area under the curve directly.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```
## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate reciever operator characteristics
roc.curve(actual, response, indices = indices)
```

Usage

```
## Generic S3 method
## for Reciever Operator Characteristics
roc.curve(...)
```

```
## Generic S3 method
## for weighted Receiver Operator Characteristics
weighted.roc.curve(...)
```

Arguments

... Arguments passed on to `roc.curve.factor`, `weighted.roc.curve.factor`

actual A vector `length` n , and k levels. Can be of `integer` or `factor`.

response A $n \times k$ `<double>`-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.

indices An optional $n \times k$ matrix of `<integer>` values of sorted response probability indices.

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

w A `<double>` vector of sample weights.

Value

A `data.frame` on the following form,

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

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`,

```
huberloss(), jaccard(), logloss(), maape(), mae(), mape(), mcc(), mpe(), mse(), nlr(),
npv(), pinball(), plr(), pr.curve(), precision(), rae(), recall(), relative.entropy(),
rmse(), rmsle(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(), zerooneloss()
```

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
  rbeta(sum(actual_classes == "Kebab"), 2, 5),
  rbeta(sum(actual_classes == "Falafel"), 5, 2)
)

## Construct response
## matrix
probability_matrix <- cbind(
  response_probabilities,
  1 - response_probabilities
)

## Visualize reciever operator characteristics

plot(
  SLmetrics::roc.curve(
    actual = actual_classes,
    response = probability_matrix
  )
)
```

Description

A generic S3 function to compute the *reciever operator characteristics* score for a classification model. This function dispatches to S3 methods in `roc.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `roc.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `roc.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_roc.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  roc.curve(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Area under the curve:

Use `auc.roc.curve` for calculating the area under the curve directly.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```
## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate reciever operator characteristics
roc.curve(actual, response, indices = indices)
```

Usage

```
## S3 method for class 'factor'
roc.curve(actual, response, thresholds = NULL, indices = NULL, ...)
```

Arguments

`actual` A vector `length` n , and k levels. Can be of `integer` or `factor`.

response	A $n \times k$ <double>-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.
thresholds	An optional <double> vector of length n (default: NULL).
indices	An optional $n \times k$ matrix of <integer> values of sorted response probability indices.
...	Arguments passed into other methods.

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

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")
```

```

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
  rbeta(sum(actual_classes == "Kebab"), 2, 5),
  rbeta(sum(actual_classes == "Falafel"), 5, 2)
)

## Construct response
## matrix
probability_matrix <- cbind(
  response_probabilities,
  1 - response_probabilities
)

## Visualize

plot(
  SLmetrics::roc.curve(
    actual = actual_classes,
    response = probability_matrix
  )
)

```

rrmse

Relative Root Mean Squared Error

Description

A generic S3 function to compute the *relative root mean squared error* score for a regression model. This function dispatches to S3 methods in `rrmse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rrmse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rrmse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rrmse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rrmse(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Relative Root Mean Squared Error
rrmse(...)

## Generic S3 method
## for weighted Concordance Correlation Coefficient
weighted_rrmse(...)
```

Arguments

... Arguments passed on to `rrmse.numeric`, `weighted_rrmse.numeric`

actual, predicted A pair of `<double>` vectors of `length` `n`.

normalization A `<double>`-value of `length` 1 (default: 1). 0: `mean`-normalization, 1: `range`-normalization, 2: `IQR`-normalization.

w A `<double>` vector of sample weights.

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone.loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rrmse(
  actual = actual_values,
  predicted = predicted_values
)
```

rrmse.numeric

*Relative Root Mean Squared Error***Description**

A generic S3 function to compute the *relative root mean squared error* score for a regression model. This function dispatches to S3 methods in `rrmse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rrmse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rrmse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rrmse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
```

```

    )
    rrmse(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

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

```

Arguments

actual, predicted A pair of `<double>` vectors of **length** *n*.

normalization A `<double>`-value of **length** 1 (default: 1). 0: **mean**-normalization, 1: **range**-normalization, 2: **IQR**-normalization.

... Arguments passed into other methods

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rrmse(
  actual = actual_values,
  predicted = predicted_values
)
```

rrse

Root Relative Squared Error

Description

A generic S3 function to compute the *root relative squared error* score for a regression model. This function dispatches to S3 methods in `rrse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rrse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rrse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rrse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rrse(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Root Relative Squared Error
rrse(...)

## Generic S3 method
## for weighted Root Relative Squared Error
weighted.rrse(...)
```

Arguments

... Arguments passed on to `rrse.numeric`, `weighted.rrse.numeric`
 actual, predicted A pair of `<double>` vectors of length n .
 w A `<double>` vector of sample weights.

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone.loss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rrse(
  actual    = actual_values,
  predicted = predicted_values
)
```

rrse.numeric	Root Relative Squared Error
--------------	-----------------------------

Description

A generic S3 function to compute the *root relative squared error* score for a regression model. This function dispatches to S3 methods in `rrse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rrse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rrse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rrse <- function(x, y, ...) {  
  stopifnot(  
    !anyNA(x), !anyNA(y),  
    length(x) == length(y)  
  )  
  rrse(x, y, ...)  
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted	A pair of <code><double></code> vectors of <code>length</code> <i>n</i> .
...	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rrse(
  actual = actual_values,
  predicted = predicted_values
)
```

rsq

 r^2

Description

A generic S3 function to compute the r^2 score for a regression model. This function dispatches to S3 methods in `rsq()` and performs no input validation. If you supply NA values or vectors of unequal length (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rsq()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rsq()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rsqu <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rsq(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for \eqn{r^2}
rsq(...)

## Generic S3 method
## for weighted \eqn{r^2}
weighted.rsq(...)
```

Arguments

... Arguments passed on to `rsq.numeric`, `weighted.rsq.numeric`

actual, predicted A pair of `<double>` vectors of `length` n .

k A `<double>`-vector of `length` 1 (default: 0). For adjusted R^2 set $k = \kappa - 1$, where κ is the number of parameters.

w A `<double>` vector of sample weights.

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rsq(
  actual = actual_values,
  predicted = predicted_values
)
```

rsq.numeric

 r^2

Description

A generic S3 function to compute the r^2 score for a regression model. This function dispatches to S3 methods in `rsq()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rsq()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rsq()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rsqu <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
}
```

```

    )
    rsq(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## S3 method for class 'numeric'
rsq(actual, predicted, k = 0, ...)

```

Arguments

actual, predicted	A pair of <code><double></code> vectors of <code>length</code> n .
k	A <code><double></code> -vector of <code>length</code> 1 (default: 0). For adjusted R^2 set $k = \kappa - 1$, where κ is the number of parameters.
...	Arguments passed into other methods

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::rsq(
  actual = actual_values,
  predicted = predicted_values
)
```

shannon.entropy

Shannon Entropy

Description

A generic S3 function to compute the *shannon entropy* score for a classification model. This function dispatches to S3 methods in `shannon.entropy()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `shannon.entropy()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `shannon.entropy()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_shannon.entropy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  shannon.entropy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## Generic S3 method
## for Shannon Entropy
shannon.entropy(...)
```

Arguments

- ... Arguments passed on to `shannon.entropy.matrix`
- `pk` A $n \times k$ `<double>`-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 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).
- `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.

Value

A `<double>` value or vector:

- A single `<double>` value (length 1) if `dim == 0`.
- A `<double>` vector with length equal to the `length` of columns if `dim == 1`.
- A `<double>` vector with length equal to the `length` of rows if `dim == 2`.

References

- MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.
- Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `specificity()`, `zerooneerror()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`, `specificity()`, `zerooneerror()`

Other Entropy: `cross.entropy()`, `logloss()`, `relative.entropy()`

Examples

```
## generate valid probability
## distributions
rand.sum <- function(n) {
  x <- sort(runif( n-1 ))
  c(x,1) - c(0, x)
}

## empirical and
## predicted probabilities
set.seed(1903)
pk <- t(replicate(200,rand.sum(5)))

## entropy
SLmetrics::shannon.entropy(
  pk = pk
)
```

shannon.entropy.matrix

Shannon Entropy

Description

A generic S3 function to compute the *shannon entropy* score for a classification model. This function dispatches to S3 methods in [shannon.entropy\(\)](#) and performs no input validation. If you supply [NA](#) values or vectors of unequal [length](#) (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because [shannon.entropy\(\)](#) operates on raw pointers, pointer-level faults (e.g. from [NA](#) or mismatched [length](#)) occur before any R-level error handling. Wrapping calls in [try\(\)](#) or [tryCatch\(\)](#) will *not* prevent R-session crashes.

To guard against this, wrap [shannon.entropy\(\)](#) in a "safe" validator that checks for [NA](#) values and matching [length](#), for example:

```
safe_shannon.entropy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  shannon.entropy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## S3 method for class 'matrix'
shannon.entropy(pk, dim = 0L, normalize = FALSE, ...)
```

Arguments

pk	A $n \times k$ <double>-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 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).
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.

Value

A <double> value or vector:

- A single <double> value (length 1) if `dim == 0`.
- A <double> vector with length equal to the [length](#) of columns if `dim == 1`.
- A <double> vector with length equal to the [length](#) of rows if `dim == 2`.

References

MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.

Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#),

npv(), pinball(), plr(), pr.curve(), precision(), rae(), recall(), relative.entropy(), rmse(), rmsle(), roc.curve(), rrmse(), rrse(), rsq(), smape(), specificity(), zerooneloss()
 Other Entropy: cross.entropy(), logloss(), relative.entropy()

Examples

```
## generate valid probability
## distributions
rand.sum <- function(n) {
  x <- sort(runif( n-1 ))
  c(x,1) - c(0, x)
}

## empirical and
## predicted probabilities
set.seed(1903)
pk <- t(replicate(200,rand.sum(5)))

## entropy
SLmetrics::shannon.entropy(
  pk = pk
)
```

smape

Symmetric Mean Absolutte Percentage Error

Description

A generic S3 function to compute the *symmetric mean absolutte percentage error* score for a regression model. This function dispatches to S3 methods in `smape()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `smape()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `smape()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_smape <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
```

```

    length(x) == length(y)
  )
  smape(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## Generic S3 method
## for Symmetric Mean Absolutte Percentage Error
smape(...)

## Generic S3 method
## for weighted Symmetric Mean Absolutte Percentage Error
weighted.smape(...)

```

Arguments

... Arguments passed on to `smape.numeric`, `weighted.smape.numeric`
 actual, predicted A pair of <double> vectors of length *n*.
 w A <double> vector of sample weights.

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::smape(
  actual = actual_values,
  predicted = predicted_values
)
```

smape.numeric

Symmetric Mean Absolutte Percentage Error

Description

A generic S3 function to compute the *symmetric mean absolute percentage error* score for a regression model. This function dispatches to S3 methods in `smape()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `smape()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `smape()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_smape <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  smape(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted
 A pair of [<double>](#) vectors of [length](#) *n*.
 ... Arguments passed into other methods

Value

A [<double>](#) value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneerror\(\)](#)

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Evaluate performance
SLmetrics::smape(
  actual    = actual_values,
  predicted = predicted_values
)
```

specificity

*Specificity***Description**

A generic S3 function to compute the *specificity* score for a classification model. This function dispatches to S3 methods in `specificity()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `specificity()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `specificity()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_specificity <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  specificity(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate specificity
## via S3 dispatching
specificity(confusion_matrix)

## additional performance metrics
## below
```

The `specificity.factor()` method calls `cmatrix()` internally, so explicitly invoking `specificity.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Specificity
specificity(...)

## Generic S3 method
## for weighted Specificity
weighted.specificity(...)
```

Arguments

... Arguments passed on to `specificity.factor`, `weighted.specificity.factor`, `specificity.cmatrix`

actual, predicted A pair of `<integer>` or `<factor>` vectors of length n , and k levels.

estimator An `<integer>`-value of length 1 (default: 0).

- 0 - a named `<double>`-vector of length k (class-wise)
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

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))`.

w A `<double>` vector of sample weights.

x A confusion matrix created `cmatrix()`.

Value

If estimator is given as

- 0 - a named `<double>` vector of length k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The specificity has other names depending on research field:

- True Negative Rate, `tnr()`
- Selectivity, `selectivity()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::specificity(
  actual = actual_classes,
  predicted = predicted_classes
)
```

specificity.cmatrix	<i>Specificity</i>
---------------------	--------------------

Description

A generic S3 function to compute the *specificity* score for a classification model. This function dispatches to S3 methods in `specificity()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `specificity()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `specificity()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_specificity <- function(x, y, ...) {  
  stopifnot(  
    !anyNA(x), !anyNA(y),  
    length(x) == length(y)  
  )  
  specificity(x, y, ...)  
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix  
confusion_matrix <- cmatrix(actual, predicted)  
  
## evaluate specificity  
## via S3 dispatching  
specificity(confusion_matrix)  
  
## additional performance metrics  
## below
```

The `specificity.factor()` method calls `cmatrix()` internally, so explicitly invoking `specificity.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
specificity(x, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

x	A confusion matrix created <code>cmatrix()</code> .
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The specificity has other names depending on research field:

- True Negative Rate, `tnr()`
- Selectivity, `selectivity()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual    = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::specificity(confusion_matrix)
```

Description

A generic S3 function to compute the *specificity* score for a classification model. This function dispatches to S3 methods in `specificity()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `specificity()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `specificity()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_specificity <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  specificity(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate specificity
## via S3 dispatching
specificity(confusion_matrix)

## additional performance metrics
## below
```

The `specificity.factor()` method calls `cmatrix()` internally, so explicitly invoking `specificity.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
specificity(actual, predicted, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The specificity has other names depending on research field:

- True Negative Rate, `tnr()`
- Selectivity, `selectivity()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`,

```
deviance.tweedie(), dor(), fbeta(), fdr(), fer(), fmi(), fpr(), gmse(), hammingloss(),
huberloss(), jaccard(), logloss(), maape(), mae(), mape(), mcc(), mpe(), mse(), nlr(),
npv(), pinball(), plr(), pr.curve(), precision(), rae(), recall(), relative.entropy(),
rmse(), rmsle(), roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), zerooneloss()
```

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::specificity(
  actual = actual_classes,
  predicted = predicted_classes
)
```

```
weighted.accuracy.factor
```

Accuracy

Description

A generic S3 function to compute the *accuracy* score for a classification model. This function dispatches to S3 methods in `accuracy()` and performs no input validation. If you supply `NA` values or vectors of unequal *length* (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `accuracy()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched *length*) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `accuracy()` in a "safe" validator that checks for `NA` values and matching *length*, for example:

```
safe_accuracy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  accuracy(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate accuracy
## via S3 dispatching
accuracy(confusion_matrix)

## additional performance metrics
## below
```

The `accuracy.factor()` method calls `cmatrix()` internally, so explicitly invoking `accuracy.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of length n , and k levels.
w	A <code><double></code> vector of sample weights.
...	Arguments passed into other methods.

Value

A `<double>`-value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.accuracy(
```

```

    actual    = actual_classes,
    predicted = predicted_classes,
    w         = sample_weights
  )

```

```
weighted.auc.pr.curve.factor
```

Area under the Precision Recall Curve

Description

A generic S3 function to compute the *area under the precision recall curve* score for a classification model. This function dispatches to S3 methods in `auc.pr.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `auc.pr.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `auc.pr.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_auc.pr.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  auc.pr.curve(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Visualizing area under the precision recall curve:

Use `pr.curve()` to construct the `data.frame` and use `plot` to visualize the area under the curve.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```

## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate area under the precision recall curve
auc.pr.curve(actual, response, indices = indices)

```

Usage

```
## S3 method for class 'factor'
weighted.auc.pr.curve(
  actual,
  response,
  w,
  estimator = 0L,
  method = 0L,
  indices = NULL,
  ...
)
```

Arguments

actual	A vector length n , and k levels. Can be of integer or factor .
response	A $n \times k$ <double> -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.
w	A <double> vector of sample weights.
estimator	An <integer> -value of length 1 (default: 0). • 0 - a named <double>-vector of length k (class-wise) • 1 - a <double> value (Micro averaged metric) • 2 - a <double> value (Macro averaged metric)
method	A <double> 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.
indices	An optional $n \times k$ matrix of <integer> values of sorted response probability indices.
...	Arguments passed into other methods.

Value

If estimator is given as

- 0: a named **<double>**-vector of **length** k
- 1: a **<double>** value (Micro averaged metric)
- 2: a **<double>** value (Macro averaged metric)

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
  rbeta(sum(actual_classes == "Kebab"), 2, 5),
  rbeta(sum(actual_classes == "Falafel"), 5, 2)
)

## Construct response
## matrix
probability_matrix <- cbind(
  response_probabilities,
  1 - response_probabilities
)

sample_weights <- runif(1e2)

## Evaluate performance

SLmetrics::weighted.auc.pr.curve(
  actual = actual_classes,
  response = probability_matrix,
```

```

      w      = sample_weights
    )

```

```
weighted.auc.roc.curve.factor
```

Area under the Receiver Operator Characteristics Curve

Description

A generic S3 function to compute the *area under the receiver operator characteristics curve* score for a classification model. This function dispatches to S3 methods in `auc.roc.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `auc.roc.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `auc.roc.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_auc.roc.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  auc.roc.curve(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Visualizing area under the receiver operator characteristics curve:

Use `roc.curve()` to construct the `data.frame` and use `plot` to visualize the area under the curve.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```

## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate area under the receiver operator characteristics curve
auc.roc.curve(actual, response, indices = indices)

```

Usage

```
## S3 method for class 'factor'
weighted.auc.roc.curve(
  actual,
  response,
  w,
  estimator = 0L,
  method = 0L,
  indices = NULL,
  ...
)
```

Arguments

actual	A vector length n , and k levels. Can be of integer or factor .
response	A $n \times k$ <double> -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.
w	A <double> vector of sample weights.
estimator	An <integer> -value of length 1 (default: 0). • 0 - a named <double>-vector of length k (class-wise) • 1 - a <double> value (Micro averaged metric) • 2 - a <double> value (Macro averaged metric)
method	A <double> 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.
indices	An optional $n \times k$ matrix of <integer> values of sorted response probability indices.
...	Arguments passed into other methods.

Value

If estimator is given as

- 0: a named **<double>**-vector of **length** k
- 1: a **<double>** value (Micro averaged metric)
- 2: a **<double>** value (Macro averaged metric)

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
  rbeta(sum(actual_classes == "Kebab"), 2, 5),
  rbeta(sum(actual_classes == "Falafel"), 5, 2)
)

## Construct response
## matrix
probability_matrix <- cbind(
  response_probabilities,
  1 - response_probabilities
)

sample_weights <- runif(1e2)

## Evaluate performance

SLmetrics::weighted.auc.roc.curve(
  actual = actual_classes,
  response = probability_matrix,
```

```

      w      = sample_weights
    )

```

```
weighted.baccuracy.factor
```

Balanced Accuracy

Description

A generic S3 function to compute the *balanced accuracy* score for a classification model. This function dispatches to S3 methods in `baccuracy()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `baccuracy()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `baccuracy()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_baccuracy <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  baccuracy(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```

## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate balanced accuracy
## via S3 dispatching
baccuracy(confusion_matrix)

## additional performance metrics
## below

```

The `baccracy.factor()` method calls `cmatrix()` internally, so explicitly invoking `baccracy.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.baccracy(actual, predicted, w, adjust = FALSE, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
w	A <code><double></code> vector of sample weights.
adjust	A <code><logical></code> value (default: <code>FALSE</code>). If <code>TRUE</code> the metric is adjusted for random chance $\frac{1}{k}$.
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.baccuracy(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)
```

weighted.brier.score.matrix

Brier Score

Description

A generic S3 function to compute the *brier score* score for a classification model. This function dispatches to S3 methods in `brier.score()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `brier.score()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `brier.score()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_brier.score <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  brier.score(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## S3 method for class 'matrix'
weighted.brier.score(ok, qk, w, ...)
```

Arguments

<code>ok</code>	A <code><double></code> indicator matrix with n samples and k classes.
<code>qk</code>	A $n \times k$ <code><double></code> -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.
<code>w</code>	A <code><double></code> vector of sample weights.
<code>...</code>	Arguments passed into other methods.

Value

A `<double>`-value

References

Gneiting, Tilmann, and Adrian E. Raftery. "Strictly proper scoring rules, prediction, and estimation." *Journal of the American statistical Association* 102.477 (2007): 359-378.

James, Gareth, et al. *An introduction to statistical learning*. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." *the Journal of machine Learning research* 12 (2011): 2825-2830.

See Also

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

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

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

## The general setup
## with 3 classes
n_obs    <- 10
n_classes <- 3

## Generate indicator matrix
## with observed outcome (ok) and
## its predicted probability matrix (qk)
ok <- diag(n_classes)[ sample.int(n_classes, n_obs, TRUE), ]
qk <- matrix(runif(n_obs * n_classes), n_obs, n_classes)
qk <- qk / rowSums(qk)

## Generate sample
## weights
sample_weights <- runif(
  n = n_obs
)

## Evaluate performance
SLmetrics::weighted.brier.score(
  ok = ok,
  qk = qk,
  w  = sample_weights
)
```

Description

A generic S3 function to compute the *concordance correlation coefficient* score for a regression model. This function dispatches to S3 methods in `ccc()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `ccc()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `ccc()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_ccc <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  ccc(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## S3 method for class 'numeric'
weighted.ccc(actual, predicted, w, correction = FALSE, ...)
```

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length</code> n .
<code>w</code>	A <code><double></code> vector of sample weights.
<code>correction</code>	A <code><logical></code> vector of <code>length</code> 1 (default: <code>FALSE</code>). If <code>TRUE</code> the variance and covariance will be adjusted with $\frac{1-n}{n}$
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.ccc(
  actual    = actual_values,
  predicted = predicted_values,
  w         = sample_weights
)
```

weighted.ckappa.factor

Cohen's κ -Statistic

Description

A generic S3 function to compute the *cohen's κ -statistic* score for a classification model. This function dispatches to S3 methods in [ckappa\(\)](#) and performs no input validation. If you supply NA values or vectors of unequal [length](#) (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `ckappa()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `ckappa()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_ckappa <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  ckappa(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate cohen's \eqn{\kappa}-statistic
## via S3 dispatching
ckappa(confusion_matrix)

## additional performance metrics
## below
```

The `ckappa.factor()` method calls `cmatrix()` internally, so explicitly invoking `ckappa.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
w	A <code><double></code> vector of sample weights.
beta	A <code><double></code> value of <code>length</code> 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.

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
```

```

sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.ckappa(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)

```

```
weighted.cmatrix.factor
```

Confusion Matrix

Description

A generic S3 function to compute the *confusion matrix* for a classification model. This function dispatches to S3 methods in `cmatrix()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `cmatrix()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `cmatrix()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_cmatrix <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  cmatrix(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

The workhorse:

`cmatrix()` is the main function for classification metrics with `cmatrix` S3 dispatch. These functions internally call `cmatrix()`, so there is a significant gain in computing the confusion matrix first, and then pass it onto the metrics. For example:

```

## Compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

```

```
## Evaluate accuracy
## via S3 dispatching
accuracy(confusion_matrix)

## Evaluate recall
## via S3 dispatching
recall(confusion_matrix)
```

Usage

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

Arguments

actual, predicted A pair of [integer](#) or [factor](#) vectors of [length](#) n , and k levels.

w A [double](#) vector of sample weights.

... Arguments passed into other methods.

Value

A named $k \times k$ [matrix](#)

Dimensions

There is no robust defensive measure against misspecifying the confusion matrix. If the arguments are passed correctly, the resulting confusion matrix is on the form:

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

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Compute confusion matrix
SLmetrics::weighted.cmatrix(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)
```

Description

A generic S3 function to compute the *gamma deviance* score for a regression model. This function dispatches to S3 methods in `deviance.gamma()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `deviance.gamma()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `deviance.gamma()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_deviance.gamma <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  deviance.gamma(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## S3 method for class 'numeric'
weighted.deviance.gamma(actual, predicted, w, ...)
```

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length</code> <i>n</i> .
<code>w</code>	A <code><double></code> vector of sample weights.
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.deviance.gamma(
  actual = actual_values,
  predicted = predicted_values,
  w = sample_weights
)
```

weighted.deviance.poisson.numeric

Poisson Deviance

Description

A generic S3 function to compute the *poisson deviance* score for a regression model. This function dispatches to S3 methods in `deviance.poisson()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `deviance.poisson()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `deviance.poisson()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_deviance.poisson <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  deviance.poisson(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## S3 method for class 'numeric'
weighted.deviance.poisson(actual, predicted, w, ...)
```

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length</code> <i>n</i> .
<code>w</code>	A <code><double></code> vector of sample weights.
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.tweedie()`,

```
dor(), fbeta(), fdr(), fer(), fmi(), fpr(), gmse(), hammingloss(), huberloss(), jaccard(),
logloss(), maape(), mae(), mape(), mcc(), mpe(), mse(), nlr(), npv(), pinball(), plr(),
pr.curve(), precision(), rae(), recall(), relative.entropy(), rmse(), rmsle(), roc.curve(),
rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(), zerooneerror()
```

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.deviance.poisson(
  actual = actual_values,
  predicted = predicted_values,
  w = sample_weights
)
```

```
weighted.deviance.tweedie.numeric
```

Tweedie Deviance

Description

A generic S3 function to compute the *tweedie deviance* score for a regression model. This function dispatches to S3 methods in `deviance.tweedie()` and performs no input validation. If you supply **NA** values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `deviance.tweedie()` operates on raw pointers, pointer-level faults (e.g. from **NA** or mismatched **length**) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `deviance.tweedie()` in a "safe" validator that checks for **NA** values and matching **length**, for example:

```
safe_deviance.tweedie <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  deviance.tweedie(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```
## S3 method for class 'numeric'
weighted.deviance.tweedie(actual, predicted, w, power = 2, ...)
```

Arguments

actual, predicted	A pair of <code><double></code> vectors of <code>length</code> n .
w	A <code><double></code> vector of sample weights.
power	A <code><double></code> value, default = 2. Tweedie power parameter. Either $\text{power} \leq 0$ or $\text{power} \geq 1$. The higher <i>power</i> , the less weight is given to extreme deviations between actual and predicted values. • power < 0: Extreme stable distribution. Requires: $\text{predicted} > 0$. • power = 0: Normal distribution, output corresponds to <code>mse()</code>, actual and predicted can be any real numbers. • power = 1: Poisson distribution (<code>deviance.poisson()</code>). Requires: $\text{actual} \geq 0$ and $\text{predicted} > 0$. • 1 < power < 2: Compound Poisson distribution. Requires: $\text{actual} \geq 0$ and $\text{predicted} > 0$. • power = 2: Gamma distribution (<code>deviance.gamma()</code>). Requires: $\text{actual} > 0$ and $\text{predicted} > 0$. • power = 3: Inverse Gaussian distribution. Requires: $\text{actual} > 0$ and $\text{predicted} > 0$. • otherwise: Positive stable distribution. Requires: $\text{actual} > 0$ and $\text{predicted} > 0$.
...	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneerror()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.deviance.tweedie(
  actual    = actual_values,
  predicted = predicted_values,
  w         = sample_weights
)
```

weighted.dor.factor	<i>Positive Likelihood Ratio</i>
---------------------	----------------------------------

Description

A generic S3 function to compute the *positive likelihood ratio* score for a classification model. This function dispatches to S3 methods in `plr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `plr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `plr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_plr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  plr(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate positive likelihood ratio
## via S3 dispatching
plr(confusion_matrix)

## additional performance metrics
## below
```

The `plr.factor()` method calls `cmatrix()` internally, so explicitly invoking `plr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

actual, predicted	A pair of <integer> or <factor> vectors of length n , and k levels.
w	A <double> vector of sample weights.
...	Arguments passed into other methods.

Value

A <double>-value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

The `nlr()`-function for the Negative Likelihood Ratio (LR-)

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)
```

```
## Evaluate performance
SLmetrics::weighted.dor(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)
```

weighted.fbeta.factor f_{β}

Description

A generic S3 function to compute the f_{β} score for a classification model. This function dispatches to S3 methods in `fbeta()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fbeta()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fbeta()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fbeta <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fbeta(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate  $f_{\beta}$ 
## via S3 dispatching
fbeta(confusion_matrix)
```

```
## additional performance metrics
## below
```

The `fbeta.factor()` method calls `cmatrix()` internally, so explicitly invoking `fbeta.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.fbeta(
  actual,
  predicted,
  w,
  beta = 1,
  estimator = 0L,
  na.rm = TRUE,
  ...
)
```

Arguments

<code>actual, predicted</code>	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
<code>w</code>	A <code><double></code> vector of sample weights.
<code>beta</code>	A <code><double></code> vector of <code>length</code> 1 (default: 1).
<code>estimator</code>	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
<code>na.rm</code>	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
<code>...</code>	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.fbeta(
```

```

    actual    = actual_classes,
    predicted = predicted_classes,
    w         = sample_weights
  )

```

weighted.fdr.factor	<i>False Discovery Rate</i>
---------------------	-----------------------------

Description

A generic S3 function to compute the *false discovery rate* score for a classification model. This function dispatches to S3 methods in `fdr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fdr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fdr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_fdr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fdr(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```

## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate false discovery rate
## via S3 dispatching
fdr(confusion_matrix)

## additional performance metrics
## below

```

The `fdr.factor()` method calls `cmatrix()` internally, so explicitly invoking `fdr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.fdr(actual, predicted, w, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <integer> or <factor> vectors of length n , and k levels.
w	A <double> vector of sample weights.
estimator	An <integer>-value of length 1 (default: 0). • 0 - a named <double>-vector of length k (class-wise) • 1 - a <double> value (Micro averaged metric) • 2 - a <double> value (Macro averaged metric)
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 <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named <double> vector of length k
- 1 - a <double> value (Micro averaged metric)
- 2 - a <double> value (Macro averaged metric)

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.fdr(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)
```

weighted.fer.factor	<i>False Omission Rate</i>
---------------------	----------------------------

Description

A generic S3 function to compute the *false omission rate* score for a classification model. This function dispatches to S3 methods in `fer()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fer()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fer()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fer <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fer(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate false omission rate
## via S3 dispatching
fer(confusion_matrix)

## additional performance metrics
## below
```

The `fer.factor()` method calls `cmatrix()` internally, so explicitly invoking `fer.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.fer(actual, predicted, w, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <integer> or <factor> vectors of length n , and k levels.
w	A <double> vector of sample weights.
estimator	An <integer>-value of length 1 (default: 0). • 0 - a named <double>-vector of length k (class-wise) • 1 - a <double> value (Micro averaged metric) • 2 - a <double> value (Macro averaged metric)
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 <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named <double> vector of [length](#) k
- 1 - a <double> value (Micro averaged metric)
- 2 - a <double> value (Macro averaged metric)

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#),

```
deviance.tweedie(), dor(), fbeta(), fdr(), fmi(), fpr(), gmse(), hammingloss(), huberloss(),
jaccard(), logloss(), maape(), mae(), mape(), mcc(), mpe(), mse(), nlr(), npv(), pinball(),
plr(), pr.curve(), precision(), rae(), recall(), relative.entropy(), rmse(), rmsle(),
roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(), zerooneerror()
```

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.fmi(
  actual = actual_classes,
  predicted = predicted_classes,
  w = sample_weights
)
```

weighted.fmi.factor *Fowlkes Mallows Index*

Description

A generic S3 function to compute the *fowlkes mallows index* score for a classification model. This function dispatches to S3 methods in `fmi()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fmi()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fmi()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_fmi <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fmi(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate fowlkes mallows index
## via S3 dispatching
fmi(confusion_matrix)

## additional performance metrics
## below
```

The `fmi.factor()` method calls `cmatrix()` internally, so explicitly invoking `fmi.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

<code>actual, predicted</code>	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
<code>w</code>	A <code><double></code> vector of sample weights.
<code>...</code>	Arguments passed into other methods.

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
```

```

)

## Evaluate performance
SLmetrics::weighted.fmi(
  actual    = actual_classes,
  predicted  = predicted_classes,
  w         = sample_weights
)

```

weighted.fpr.factor *False Positive Rate*

Description

A generic S3 function to compute the *false positive rate* score for a classification model. This function dispatches to S3 methods in `fpr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `fpr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `fpr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_fpr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  fpr(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```

## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate false positive rate
## via S3 dispatching
fpr(confusion_matrix)

```

```
## additional performance metrics
## below
```

The `fpr.factor()` method calls `cmatrix()` internally, so explicitly invoking `fpr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.fpr(actual, predicted, w, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
w	A <code><double></code> vector of sample weights.
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The false positive rate has other names depending on research field:

- Fallout, `fallout()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.fpr(
```

```

    actual    = actual_classes,
    predicted = predicted_classes,
    w         = sample_weights
  )

```

weighted.gmse.numeric *Geometric Mean Squared Error*

Description

A generic S3 function to compute the *geometric mean squared error* score for a regression model. This function dispatches to S3 methods in `gmse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `gmse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `gmse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_gmse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  gmse(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

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

```

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length</code> <i>n</i> .
<code>w</code>	A <code><double></code> vector of sample weights.
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.gmse(
  actual = actual_values,
  predicted = predicted_values,
  w = sample_weights
)
```

weighted.hammingloss.factor

Hamming Loss

Description

A generic S3 function to compute the *hamming loss* score for a classification model. This function dispatches to S3 methods in `hammingloss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `hammingloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `hammingloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_hammingloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  hammingloss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate hamming loss
## via S3 dispatching
hammingloss(confusion_matrix)

## additional performance metrics
## below
```

The `hammingloss.factor()` method calls `cmatrix()` internally, so explicitly invoking `hammingloss.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

actual, predicted
 A pair of [integer](#) or [factor](#) vectors of [length](#) n , and k levels.
 w
 A [double](#) vector of sample weights.
 ...
 Arguments passed into other methods.

Value

A [double](#)-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)
```

```

)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.hammingloss(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)

```

weighted.huberloss.numeric

Huber Loss

Description

A generic S3 function to compute the *huber loss* score for a regression model. This function dispatches to S3 methods in `huberloss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `huberloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `huberloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_huberloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  huberloss(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

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

```

Arguments

actual, predicted	A pair of <double> vectors of length <i>n</i> .
w	A <double> vector of sample weights.
delta	A <double>-vector of length 1 (default: 1). The threshold value for switch between functions (see calculation).
...	Arguments passed into other methods

Value

A <double> value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.huberloss(
```

```

    actual    = actual_values,
    predicted = predicted_values,
    w         = sample_weights
  )

```

 weighted.jaccard.factor

Jaccard Index

Description

A generic S3 function to compute the *jaccard index* score for a classification model. This function dispatches to S3 methods in `jaccard()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `jaccard()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `jaccard()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_jaccard <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  jaccard(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```

## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate jaccard index
## via S3 dispatching
jaccard(confusion_matrix)

## additional performance metrics
## below

```

The `jaccard.factor()` method calls `cmatrix()` internally, so explicitly invoking `jaccard.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.jaccard(actual, predicted, w, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

<code>actual, predicted</code>	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
<code>w</code>	A <code><double></code> vector of sample weights.
<code>estimator</code>	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
<code>na.rm</code>	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
<code>...</code>	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The specificity has other names depending on research field:

- Critical Success Index, `csi()`
- Threat Score, `tscore()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.jaccard(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)
```

weighted.logloss.factor

Logarithmic Loss

Description

A generic S3 function to compute the *logarithmic loss* score for a classification model. This function dispatches to S3 methods in `logloss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `logloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `logloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_logloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  logloss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual	A vector <code>length</code> n , and k levels. Can be of <code>integer</code> or <code>factor</code> .
response	A $n \times k$ <code><double></code> -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 <code>actual</code> , the second column to the second factor level, and so on.
w	A <code><double></code> vector of sample weights.
normalize	A <code><logical></code> -value (default: <code>TRUE</code>). If <code>TRUE</code> , the mean cross-entropy across all observations is returned; otherwise, the sum of cross-entropies is returned.
...	Arguments passed into other methods.

Value

A <double>

References

MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.

Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracity\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Entropy: [cross.entropy\(\)](#), [relative.entropy\(\)](#), [shannon.entropy\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted response
## probabilities
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

response <- runif(n = 1e3)

## Generate sample
## weights
sample_weights <- runif(
  n = 1e3
)
```

```
## Evaluate performance
SLmetrics::weighted.logloss(
  actual    = actual_classes,
  response  = cbind(
    response,
    1 - response
  ),
  w = sample_weights
)

## Generate observed
## frequencies
actual_frequency <- sample(10L:100L, size = 1e3, replace = TRUE)

SLmetrics::weighted.logloss(
  actual    = actual_frequency,
  response  = response,
  w        = sample_weights
)
```

weighted.logloss.integer

Logarithmic Loss

Description

A generic S3 function to compute the *logarithmic loss* score for a classification model. This function dispatches to S3 methods in `logloss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `logloss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `logloss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_logloss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  logloss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual	A vector length n , and k levels. Can be of integer or factor .
response	A $n \times k$ <double> -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.
w	A <double> vector of sample weights.
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.

Value

A **<double>**

References

MacKay, David JC. Information theory, inference and learning algorithms. Cambridge university press, 2003.

Kramer, Oliver, and Oliver Kramer. "Scikit-learn." Machine learning for evolution strategies (2016): 45-53.

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Entropy: [cross.entropy\(\)](#), [relative.entropy\(\)](#), [shannon.entropy\(\)](#)

Examples

```

## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted response
## probabilities
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

response <- runif(n = 1e3)

## Generate sample
## weights
sample_weights <- runif(
  n = 1e3
)

## Evaluate performance
SLmetrics::weighted.logloss(
  actual = actual_classes,
  response = cbind(
    response,
    1 - response
  ),
  w = sample_weights
)

## Generate observed
## frequencies
actual_frequency <- sample(10L:100L, size = 1e3, replace = TRUE)

SLmetrics::weighted.logloss(
  actual = actual_frequency,
  response = response,
  w = sample_weights
)

```

weighted.maape.numeric

Mean Arctangent Absolute Percentage Error

Description

A generic S3 function to compute the *mean arctangent absolute percentage error* score for a regression model. This function dispatches to S3 methods in `maape()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `maape()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `maape()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_maape <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  maape(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length n</code> .
<code>w</code>	A <code><double></code> vector of sample weights.
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.maape(
  actual = actual_values,
  predicted = predicted_values,
  w = sample_weights
)
```

weighted.mae.numeric *Mean Absolute Error*

Description

A generic S3 function to compute the *mean absolute error* score for a regression model. This function dispatches to S3 methods in `mae()` and performs no input validation. If you supply **NA** values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mae()` operates on raw pointers, pointer-level faults (e.g. from **NA** or mismatched **length**) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mae()` in a "safe" validator that checks for **NA** values and matching **length**, for example:

```
safe_mae <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mae(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted	A pair of <double> vectors of length <i>n</i> .
w	A <double> vector of sample weights.
...	Arguments passed into other methods

Value

A [<double>](#) value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#),

```
rmsle(), roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(),
zerooneerror()
```

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.mae(
  actual    = actual_values,
  predicted = predicted_values,
  w         = sample_weights
)
```

weighted.mape.numeric *Mean Absolute Percentage Error*

Description

A generic S3 function to compute the *mean absolute percentage error* score for a regression model. This function dispatches to S3 methods in `mape()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mape()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mape()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_mape <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mape(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

```
actual, predicted      A pair of <double> vectors of length n.
w                      A <double> vector of sample weights.
...                    Arguments passed into other methods
```

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mpe\(\)](#), [mse\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
```

```

sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.mape(
  actual    = actual_values,
  predicted = predicted_values,
  w         = sample_weights
)

```

weighted.mcc.factor *Matthews Correlation Coefficient*

Description

A generic S3 function to compute the *matthews correlation coefficient* score for a classification model. This function dispatches to S3 methods in `mcc()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mcc()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mcc()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_mcc <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mcc(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```

## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate matthews correlation coefficient
## via S3 dispatching
mcc(confusion_matrix)

```

```
## additional performance metrics
## below
```

The `mcc.factor()` method calls `cmatrix()` internally, so explicitly invoking `mcc.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

actual, predicted	A pair of <integer> or <factor> vectors of length n , and k levels.
w	A <double> vector of sample weights.
...	Arguments passed into other methods.

Value

A <double>-value

Other names

The Matthews Correlation Coefficient has other names depending on research field:

- ϕ -coefficient, `phi()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `nlr()`, `npv()`,

```
pinball(),plr(),pr.curve(),precision(),rae(),recall(),relative.entropy(),rmse(),
rmsle(),roc.curve(),rrmse(),rrse(),rsq(),shannon.entropy(),smape(),specificity(),
zerooneerror()
```

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.mcc(
  actual = actual_classes,
  predicted = predicted_classes,
  w = sample_weights
)
```

weighted.mpe.numeric *Mean Percentage Error*

Description

A generic S3 function to compute the *mean percentage error* score for a regression model. This function dispatches to S3 methods in `mpe()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mpe()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mpe()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_mpe <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mpe(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length</code> <code>n</code> .
<code>w</code>	A <code><double></code> vector of sample weights.
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.mpe(
  actual    = actual_values,
  predicted = predicted_values,
  w         = sample_weights
)
```

`weighted.mse.numeric` *Mean Squared Error*

Description

A generic S3 function to compute the *mean squared error* score for a regression model. This function dispatches to S3 methods in `mse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `mse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `mse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_mse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  mse(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted	A pair of <double> vectors of length <i>n</i> .
w	A <double> vector of sample weights.
...	Arguments passed into other methods

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#),

```
rmsle(), roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(),
zerooneerror()
```

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.mse(
  actual = actual_values,
  predicted = predicted_values,
  w = sample_weights
)
```

weighted.nlr.factor *Negative Likelihood Ratio*

Description

A generic S3 function to compute the *negative likelihood ratio* score for a classification model. This function dispatches to S3 methods in `nlr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `nlr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `nlr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_nlr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  nlr(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate negative likelihood ratio
## via S3 dispatching
nlr(confusion_matrix)

## additional performance metrics
## below
```

The `nlr.factor()` method calls `cmatrix()` internally, so explicitly invoking `nlr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of length n , and k levels.
w	A <code><double></code> vector of sample weights.
...	Arguments passed into other methods.

Value

A `<double>`-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

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

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`,

```
jaccard(), logloss(), mcc(), npv(), plr(), pr.curve(), precision(), recall(), relative.entropy(),
roc.curve(), shannon.entropy(), specificity(), zerooneloss()
```

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.nlr(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)
```

Description

A generic S3 function to compute the *negative predictive value* score for a classification model. This function dispatches to S3 methods in `npv()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `npv()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `npv()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_npv <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  npv(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate negative predictive value
## via S3 dispatching
npv(confusion_matrix)

## additional performance metrics
## below
```

The `npv.factor()` method calls `cmatrix()` internally, so explicitly invoking `npv.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.npv(actual, predicted, w, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
w	A <code><double></code> vector of sample weights.
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracity()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.npv(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)
```

weighted.pinball.numeric

Pinball Loss

Description

A generic S3 function to compute the *pinball loss* score for a regression model. This function dispatches to S3 methods in `pinball()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `pinball()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `pinball()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_pinball <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  pinball(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length</code> n .
<code>w</code>	A <code><double></code> vector of sample weights.
<code>alpha</code>	A <code><double></code> -value of <code>length</code> 1 (default: 0.5). The slope of the pinball loss function.
<code>deviance</code>	A <code><logical></code> -value of <code>length</code> 1 (default: <code>FALSE</code>). If <code>TRUE</code> the function returns the D^2 loss.
<code>...</code>	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.pinball(
  actual    = actual_values,
  predicted = predicted_values,
  w         = sample_weights
)
```

weighted.plr.factor	<i>Positive Likelihood Ratio</i>
---------------------	----------------------------------

Description

A generic S3 function to compute the *positive likelihood ratio* score for a classification model. This function dispatches to S3 methods in `plr()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `plr()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `plr()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_plr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  plr(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate positive likelihood ratio
## via S3 dispatching
plr(confusion_matrix)

## additional performance metrics
## below
```

The `plr.factor()` method calls `cmatrix()` internally, so explicitly invoking `plr.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of length n , and k levels.
w	A <code><double></code> vector of sample weights.
...	Arguments passed into other methods.

Value

A `<double>`-value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

The `nlr()`-function for the Negative Likelihood Ratio (LR-)

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)
```

```
## Evaluate performance
SLmetrics::weighted.plr(
  actual    = actual_classes,
  predicted  = predicted_classes,
  w         = sample_weights
)
```

```
weighted.pr.curve.factor
```

Precision Recall Curve

Description

A generic S3 function to compute the *precision recall curve* score for a classification model. This function dispatches to S3 methods in `pr.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `pr.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `pr.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_pr.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  pr.curve(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Area under the curve:

Use `auc.pr.curve` for calculating the area under the curve directly.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```
## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate precision recall curve
pr.curve(actual, response, indices = indices)
```

Usage

```
## S3 method for class 'factor'
weighted.pr.curve(actual, response, w, thresholds = NULL, indices = NULL, ...)
```

Arguments

actual	A vector length n , and k levels. Can be of integer or factor .
response	A $n \times k$ double -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.
w	A double vector of sample weights.
thresholds	An optional double vector of length n (default: NULL).
indices	An optional $n \times k$ matrix of integer values of sorted response probability indices.
...	Arguments passed into other methods.

Value

A [data.frame](#) on the following form,

threshold	numeric Thresholds used to determine recall() and precision()
level	character The level of the actual factor
label	character The levels of the actual factor
recall	numeric The recall
precision	numeric The precision

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
  rbeta(sum(actual_classes == "Kebab"), 2, 5),
  rbeta(sum(actual_classes == "Falafel"), 5, 2)
)

## Construct response
## matrix
probability_matrix <- cbind(
  response_probabilities,
  1 - response_probabilities
)

sample_weights <- runif(1e2)

## Visualize
plot(
  SLmetrics::weighted.pr.curve(
```

```

    actual  = actual_classes,
    response = probability_matrix,
    w       = sample_weights
  )
)
```

```
weighted.precision.factor
```

```
Precision
```

Description

A generic S3 function to compute the *precision* score for a classification model. This function dispatches to S3 methods in `precision()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `precision()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `precision()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_precision <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  precision(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate precision
## via S3 dispatching
```

```
precision(confusion_matrix)
```

```
## additional performance metrics
## below
```

The `precision.factor()` method calls `cmatrix()` internally, so explicitly invoking `precision.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.precision(actual, predicted, w, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <integer> or <factor> vectors of length n , and k levels.
w	A <double> vector of sample weights.
estimator	An <integer>-value of length 1 (default: 0). • 0 - a named <double>-vector of length k (class-wise) • 1 - a <double> value (Micro averaged metric) • 2 - a <double> value (Macro averaged metric)
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 <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named <double> vector of length k
- 1 - a <double> value (Micro averaged metric)
- 2 - a <double> value (Macro averaged metric)

Other names

The precision has other names depending on research field:

- Positive Predictive Value, `ppv()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.precision(
```

```

    actual    = actual_classes,
    predicted = predicted_classes,
    w         = sample_weights
  )

```

weighted.rae.numeric *Relative Absolute Error*

Description

A generic S3 function to compute the *relative absolute error* score for a regression model. This function dispatches to S3 methods in `rae()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rae()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rae()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_rae <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rae(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

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

```

Arguments

actual, predicted	A pair of <code><double></code> vectors of <code>length</code> <i>n</i> .
w	A <code><double></code> vector of sample weights.
...	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneerror()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.rae(
  actual    = actual_values,
  predicted = predicted_values,
  w         = sample_weights
)
```

weighted.recall.factor

Recall

Description

A generic S3 function to compute the *recall* score for a classification model. This function dispatches to S3 methods in `recall()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `recall()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `recall()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_recall <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  recall(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate recall
## via S3 dispatching
recall(confusion_matrix)

## additional performance metrics
## below
```

The `recall.factor()` method calls `cmatrix()` internally, so explicitly invoking `recall.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.recall(actual, predicted, w, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <integer> or <factor> vectors of length n , and k levels.
w	A <double> vector of sample weights.
estimator	An <integer>-value of length 1 (default: 0). • 0 - a named <double>-vector of length k (class-wise) • 1 - a <double> value (Micro averaged metric) • 2 - a <double> value (Macro averaged metric)
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 <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named <double> vector of [length](#) k
- 1 - a <double> value (Micro averaged metric)
- 2 - a <double> value (Macro averaged metric)

Other names

The Recall has other names depending on research field:

- Sensitivity, [sensitivity\(\)](#)
- True Positive Rate, [tpr\(\)](#)

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#), [zerooneloss\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.recall(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)
```

weighted.rmse.numeric *Root Mean Squared Error*

Description

A generic S3 function to compute the *root mean squared error* score for a regression model. This function dispatches to S3 methods in [rmse\(\)](#) and performs no input validation. If you supply [NA](#)

values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rmse()` operates on raw pointers, pointer-level faults (e.g. from **NA** or mismatched **length**) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rmse()` in a "safe" validator that checks for **NA** values and matching **length**, for example:

```
safe_rmse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rmse(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

<code>actual, predicted</code>	A pair of <double> vectors of length <i>n</i> .
<code>w</code>	A <double> vector of sample weights.
<code>...</code>	Arguments passed into other methods

Value

A **<double>** value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.rmse(
  actual = actual_values,
  predicted = predicted_values,
  w = sample_weights
)
```

weighted.rmsle.numeric

Root Mean Squared Logarithmic Error

Description

A generic S3 function to compute the *root mean squared logarithmic error* score for a regression model. This function dispatches to S3 methods in `rmsle()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rmsle()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rmsle()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rmsle <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rmsle(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted	A pair of <double> vectors of length <i>n</i> .
w	A <double> vector of sample weights.
...	Arguments passed into other methods

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#),

```
rmse(), roc.curve(), rrmse(), rrse(), rsq(), shannon.entropy(), smape(), specificity(),
zerooneerror()
```

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.rmsle(
  actual = actual_values,
  predicted = predicted_values,
  w = sample_weights
)
```

weighted.roc.curve.factor

Reciever Operator Characteristics

Description

A generic S3 function to compute the *reciever operator characteristics* score for a classification model. This function dispatches to S3 methods in `roc.curve()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `roc.curve()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `roc.curve()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_roc.curve <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  roc.curve(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Area under the curve:

Use [auc.roc.curve](#) for calculating the area under the curve directly.

Efficient multi-metric evaluation:

To avoid sorting the same probability matrix multiple times (once per class or curve), you can precompute a single set of sort indices and pass it via the `indices` argument. This reduces the overall cost from $O(K \cdot N \log N)$ to $O(N \log N + K \cdot N)$.

```
## presort response
## probabilities
indices <- preorder(response, decreasing = TRUE)

## evaluate receiver operator characteristics
roc.curve(actual, response, indices = indices)
```

Usage

```
## S3 method for class 'factor'
weighted.roc.curve(actual, response, w, thresholds = NULL, indices = NULL, ...)
```

Arguments

<code>actual</code>	A vector length n , and k levels. Can be of integer or factor .
<code>response</code>	A $n \times k$ double -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 <code>actual</code> , the second column to the second factor level, and so on.
<code>w</code>	A double vector of sample weights.
<code>thresholds</code>	An optional double vector of length n (default: NULL).
<code>indices</code>	An optional $n \times k$ matrix of integer values of sorted response probability indices.
<code>...</code>	Arguments passed into other methods.

Value

A [data.frame](#) on the following form,

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

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `shannon.entropy()`, `specificity()`, `zerooneloss()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zerooneloss()`

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual classes
## and response probabilities
actual_classes <- factor(
  x = sample(
    x = classes,
    size = 1e2,
    replace = TRUE,
    prob = c(0.7, 0.3)
  )
)

response_probabilities <- ifelse(
  actual_classes == "Kebab",
  rbeta(sum(actual_classes == "Kebab"), 2, 5),
  rbeta(sum(actual_classes == "Falafel"), 5, 2)
)

## Construct response
## matrix
probability_matrix <- cbind(
  response_probabilities,
  1 - response_probabilities
)
```

```

)

sample_weights <- runif(1e2)

## Visualize

plot(
  SLmetrics::weighted.roc.curve(
    actual = actual_classes,
    response = probability_matrix,
    w      = sample_weights
  )
)

```

weighted.rrmse.numeric

Relative Root Mean Squared Error

Description

A generic S3 function to compute the *relative root mean squared error* score for a regression model. This function dispatches to S3 methods in `rrmse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rrmse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rrmse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_rrmse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rrmse(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted A pair of `<double>` vectors of `length` n .

w A `<double>` vector of sample weights.

normalization A `<double>`-value of `length` 1 (default: 1). 0: `mean`-normalization, 1: `range`-normalization, 2: `IQR`-normalization.

... Arguments passed into other methods

Value

A `<double>` value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrse()`, `rsq()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrse()`, `rsq()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)
```

```
## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.rrmse(
  actual    = actual_values,
  predicted = predicted_values,
  w         = sample_weights
)
```

weighted.rrse.numeric *Root Relative Squared Error*

Description

A generic S3 function to compute the *root relative squared error* score for a regression model. This function dispatches to S3 methods in `rrse()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rrse()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rrse()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_rrse <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rrse(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted
 A pair of <double> vectors of [length](#) *n*.
 w A <double> vector of sample weights.
 ... Arguments passed into other methods

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: [ccc\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [gmse\(\)](#), [huberloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mpe\(\)](#), [mse\(\)](#), [pinball\(\)](#), [rae\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [rrmse\(\)](#), [rsq\(\)](#), [smape\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#), [zeroone loss\(\)](#)

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.rrse(
  actual    = actual_values,
```

```

    predicted = predicted_values,
    w          = sample_weights
  )

```

```
weighted.rsq.numeric   $r^2$ 
```

Description

A generic S3 function to compute the r^2 score for a regression model. This function dispatches to S3 methods in `rsq()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `rsq()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `rsq()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```

safe_rsqr <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  rsq(x, y, ...)
}

```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

```

## S3 method for class 'numeric'
weighted.rsq(actual, predicted, w, k = 0, ...)

```

Arguments

<code>actual, predicted</code>	A pair of <code><double></code> vectors of <code>length</code> n .
<code>w</code>	A <code><double></code> vector of sample weights.
<code>k</code>	A <code><double></code> -vector of <code>length</code> 1 (default: 0). For adjusted R^2 set $k = \kappa - 1$, where κ is the number of parameters.
<code>...</code>	Arguments passed into other methods

Value

A <double> value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `smape()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `shannon.entropy()`, `smape()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.rsq(
  actual    = actual_values,
  predicted = predicted_values,
  w         = sample_weights
)
```

weighted.smape.numeric

Symmetric Mean Absolutte Percentage Error

Description

A generic S3 function to compute the *symmetric mean absolute percentage error* score for a regression model. This function dispatches to S3 methods in `smape()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `smape()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `smape()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_smape <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  smape(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Usage

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

Arguments

actual, predicted	A pair of <code><double></code> vectors of <code>length</code> <i>n</i> .
w	A <code><double></code> vector of sample weights.
...	Arguments passed into other methods

Value

A `<double>` value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Virtanen, Pauli, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python." Nature methods 17.3 (2020): 261-272.
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Regression: `ccc()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `gmse()`, `huberloss()`, `maape()`, `mae()`, `mape()`, `mpe()`, `mse()`, `pinball()`, `rae()`, `rmse()`, `rmsle()`, `rrmse()`, `rrse()`, `rsq()`

Other Supervised Learning: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ccc()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `deviance.gamma()`, `deviance.poisson()`, `deviance.tweedie()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `gmse()`, `hammingloss()`, `huberloss()`, `jaccard()`, `logloss()`, `maape()`, `mae()`, `mape()`, `mcc()`, `mpe()`, `mse()`, `nlr()`, `npv()`, `pinball()`, `plr()`, `pr.curve()`, `precision()`, `rae()`, `recall()`, `relative.entropy()`, `rmse()`, `rmsle()`, `roc.curve()`, `rrmse()`, `rrse()`, `rsq()`, `shannon.entropy()`, `specificity()`, `zeroone loss()`

Examples

```
## Generate actual
## and predicted values
actual_values    <- c(1.3, 0.4, 1.2, 1.4, 1.9, 1.0, 1.2)
predicted_values <- c(0.7, 0.5, 1.1, 1.2, 1.8, 1.1, 0.2)

## Generate sample
## weights
sample_weights <- c(0.3, 0.5, 0.3, 0, 0.8, 0.8, 1)

## Evaluate performance
SLmetrics::weighted.smape(
  actual    = actual_values,
  predicted = predicted_values,
  w         = sample_weights
)
```

weighted.specificity.factor
Specificity

Description

A generic S3 function to compute the *specificity* score for a classification model. This function dispatches to S3 methods in `specificity()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `specificity()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `specificity()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_specificity <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  specificity(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate specificity
## via S3 dispatching
specificity(confusion_matrix)

## additional performance metrics
## below
```

The `specificity.factor()` method calls `cmatrix()` internally, so explicitly invoking `specificity.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.specificity(actual, predicted, w, estimator = 0L, na.rm = TRUE, ...)
```

Arguments

actual, predicted	A pair of <code><integer></code> or <code><factor></code> vectors of <code>length</code> n , and k levels.
w	A <code><double></code> vector of sample weights.
estimator	An <code><integer></code> -value of <code>length</code> 1 (default: 0). • 0 - a named <code><double></code>-vector of <code>length</code> k (class-wise) • 1 - a <code><double></code> value (Micro averaged metric) • 2 - a <code><double></code> value (Macro averaged metric)
na.rm	A <code><logical></code> value of <code>length</code> 1 (default: <code>TRUE</code>). If <code>TRUE</code> , <code>NA</code> values are removed from the computation. This argument is only relevant when <code>micro != NULL</code> . When <code>na.rm = TRUE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(na.omit(c(1, 2, NA)))</code> . When <code>na.rm = FALSE</code> , the computation corresponds to <code>sum(c(1, 2, NA), na.rm = TRUE) / length(c(1, 2, NA))</code> .
...	Arguments passed into other methods.

Value

If estimator is given as

- 0 - a named `<double>` vector of `length` k
- 1 - a `<double>` value (Micro averaged metric)
- 2 - a `<double>` value (Macro averaged metric)

Other names

The specificity has other names depending on research field:

- True Negative Rate, `tnr()`
- Selectivity, `selectivity()`

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: `accuracy()`, `auc.pr.curve()`, `auc.roc.curve()`, `baccuracy()`, `brier.score()`, `ckappa()`, `cmatrix()`, `cross.entropy()`, `dor()`, `fbeta()`, `fdr()`, `fer()`, `fmi()`, `fpr()`, `hammingloss()`, `jaccard()`, `logloss()`, `mcc()`, `nlr()`, `npv()`, `plr()`, `pr.curve()`, `precision()`, `recall()`, `relative.entropy()`, `roc.curve()`, `shannon.entropy()`, `zerooneerror()`

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [zeroone.loss\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
)

## Evaluate performance
SLmetrics::weighted.specificity(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)
```

weighted.zeroone.loss.factor

Zero-One Loss

Description

A generic S3 function to compute the *zero-one loss* score for a classification model. This function dispatches to S3 methods in [zeroone.loss\(\)](#) and performs no input validation. If you supply [NA](#)

values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `zeroone loss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched **length**) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `zeroone loss()` in a "safe" validator that checks for `NA` values and matching **length**, for example:

```
safe_zeroone loss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  zeroone loss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate zero-one loss
## via S3 dispatching
zeroone loss(confusion_matrix)

## additional performance metrics
## below
```

The `zeroone loss.factor()` method calls `cmatrix()` internally, so explicitly invoking `zeroone loss.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'factor'
weighted.zeroone loss(actual, predicted, w, ...)
```

Arguments

<code>actual, predicted</code>	A pair of <code><integer></code> or <code><factor></code> vectors of length n , and k levels.
<code>w</code>	A <code><double></code> vector of sample weights.
<code>...</code>	Arguments passed into other methods.

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Generate sample
## weights
sample_weights <- runif(
  n = length(actual_classes)
```

```

)

## Evaluate performance
SLmetrics::weighted.zerooneerror(
  actual    = actual_classes,
  predicted = predicted_classes,
  w         = sample_weights
)

```

wine.quality

Wine quality dataset

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.

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 ≥ 7 .
 - Low Quality** quality ≤ 4 .
 - Medium Quality** for all other quality scores.

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).

References

Cortez, Paulo, et al. "Modeling wine preferences by data mining from physicochemical properties." Decision support systems 47.4 (2009): 547-553.

 zeroone loss

 Zero-One Loss

Description

A generic S3 function to compute the *zero-one loss* score for a classification model. This function dispatches to S3 methods in `zeroone loss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `zeroone loss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `zeroone loss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_zeroone loss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  zeroone loss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate zero-one loss
## via S3 dispatching
zeroone loss(confusion_matrix)

## additional performance metrics
## below
```

The `zeroone loss.factor()` method calls `cmatrix()` internally, so explicitly invoking `zeroone loss.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## Generic S3 method
## for Zero-One Loss
zeroone loss(...)

## Generic S3 method
## for weighted Zero-One Loss
weighted.zeroone loss(...)
```

Arguments

... Arguments passed on to `zeroone loss.factor`, `weighted.zeroone loss.factor`, `zeroone loss.cmatrix`

actual, predicted A pair of `<integer>` or `<factor>` vectors of length n , and k levels.

w A `<double>` vector of sample weights.

x A confusion matrix created `cmatrix()`.

Value

A `<double>`-value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Evaluate performance
SLmetrics::zeroone loss(
  actual = actual_classes,
  predicted = predicted_classes
)
```

zeroone loss.cmatrix *Zero-One Loss*

Description

A generic S3 function to compute the *zero-one loss* score for a classification model. This function dispatches to S3 methods in [zeroone loss\(\)](#) and performs no input validation. If you supply **NA** values or vectors of unequal **length** (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `zeroone loss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `zeroone loss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_zeroone loss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  zeroone loss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate zero-one loss
## via S3 dispatching
zeroone loss(confusion_matrix)

## additional performance metrics
## below
```

The `zeroone loss.factor()` method calls `cmatrix()` internally, so explicitly invoking `zeroone loss.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

```
## S3 method for class 'cmatrix'
zeroone loss(x, ...)
```

Arguments

```
x          A confusion matrix created cmatrix().
...        Arguments passed into other methods.
```

Value

A `<double>`-value

References

- James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.
- Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).
- Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccuracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

predicted_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)

## Construct confusion
## matrix
confusion_matrix <- SLmetrics::cmatrix(
  actual = actual_classes,
  predicted = predicted_classes
)

## Evaluate performance
SLmetrics::zerooneLoss(confusion_matrix)
```

zeroone loss.factor	Zero-One Loss
---------------------	---------------

Description

A generic S3 function to compute the *zero-one loss* score for a classification model. This function dispatches to S3 methods in `zeroone loss()` and performs no input validation. If you supply `NA` values or vectors of unequal `length` (e.g. `length(x) != length(y)`), the underlying C++ code may trigger undefined behavior and crash your R session.

Defensive measures:

Because `zeroone loss()` operates on raw pointers, pointer-level faults (e.g. from `NA` or mismatched `length`) occur before any R-level error handling. Wrapping calls in `try()` or `tryCatch()` will *not* prevent R-session crashes.

To guard against this, wrap `zeroone loss()` in a "safe" validator that checks for `NA` values and matching `length`, for example:

```
safe_zeroone loss <- function(x, y, ...) {
  stopifnot(
    !anyNA(x), !anyNA(y),
    length(x) == length(y)
  )
  zeroone loss(x, y, ...)
}
```

Apply the same pattern to any custom metric functions to ensure input sanity before calling the underlying C++ code.

Efficient multi-metric evaluation:

For multiple performance evaluations of a classification model, first compute the confusion matrix once via `cmatrix()`. All other performance metrics can then be derived from this one object via S3 dispatching:

```
## compute confusion matrix
confusion_matrix <- cmatrix(actual, predicted)

## evaluate zero-one loss
## via S3 dispatching
zeroone loss(confusion_matrix)

## additional performance metrics
## below
```

The `zeroone loss.factor()` method calls `cmatrix()` internally, so explicitly invoking `zeroone loss.cmatrix()` yourself avoids duplicate computation, yielding significant speed and memory efficiency gains when you need multiple evaluation metrics.

Usage

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

Arguments

actual, predicted A pair of <integer> or <factor> vectors of length n , and k levels.

... Arguments passed into other methods.

Value

A <double>-value

References

James, Gareth, et al. An introduction to statistical learning. Vol. 112. No. 1. New York: springer, 2013.

Hastie, Trevor. "The elements of statistical learning: data mining, inference, and prediction." (2009).

Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." the Journal of machine Learning research 12 (2011): 2825-2830.

See Also

Other Classification: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccruracy\(\)](#), [brier.score\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [hammingloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [mcc\(\)](#), [nlr\(\)](#), [npv\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [roc.curve\(\)](#), [shannon.entropy\(\)](#), [specificity\(\)](#)

Other Supervised Learning: [accuracy\(\)](#), [auc.pr.curve\(\)](#), [auc.roc.curve\(\)](#), [baccruracy\(\)](#), [brier.score\(\)](#), [ccc\(\)](#), [ckappa\(\)](#), [cmatrix\(\)](#), [cross.entropy\(\)](#), [deviance.gamma\(\)](#), [deviance.poisson\(\)](#), [deviance.tweedie\(\)](#), [dor\(\)](#), [fbeta\(\)](#), [fdr\(\)](#), [fer\(\)](#), [fmi\(\)](#), [fpr\(\)](#), [gmse\(\)](#), [hammingloss\(\)](#), [huberloss\(\)](#), [jaccard\(\)](#), [logloss\(\)](#), [maape\(\)](#), [mae\(\)](#), [mape\(\)](#), [mcc\(\)](#), [mpe\(\)](#), [mse\(\)](#), [nlr\(\)](#), [npv\(\)](#), [pinball\(\)](#), [plr\(\)](#), [pr.curve\(\)](#), [precision\(\)](#), [rae\(\)](#), [recall\(\)](#), [relative.entropy\(\)](#), [rmse\(\)](#), [rmsle\(\)](#), [roc.curve\(\)](#), [rrmse\(\)](#), [rrse\(\)](#), [rsq\(\)](#), [shannon.entropy\(\)](#), [smape\(\)](#), [specificity\(\)](#)

Examples

```
## Classes and
## seed
set.seed(1903)
classes <- c("Kebab", "Falafel")

## Generate actual
## and predicted classes
actual_classes <- factor(
  x = sample(x = classes, size = 1e3, replace = TRUE),
  levels = c("Kebab", "Falafel")
)
```

```
predicted_classes <- factor(  
  x = sample(x = classes, size = 1e3, replace = TRUE),  
  levels = c("Kebab", "Falafel")  
)  
  
## Evaluate performance  
SLmetrics::zeroone loss(  
  actual    = actual_classes,  
  predicted = predicted_classes  
)
```

Index

* Classification

- accuracy, [5](#)
- auc.pr.curve, [12](#)
- auc.roc.curve, [18](#)
- baccuracy, [25](#)
- brier.score.matrix, [34](#)
- ckappa, [41](#)
- cmatrix, [48](#)
- cross.entropy, [53](#)
- dor, [69](#)
- fbeta, [76](#)
- fdr, [84](#)
- fer, [92](#)
- fmi, [100](#)
- fpr, [107](#)
- hammingloss, [118](#)
- jaccard, [129](#)
- logloss, [137](#)
- mcc, [154](#)
- nlr, [169](#)
- npv, [176](#)
- plr, [190](#)
- pr.curve, [197](#)
- precision, [203](#)
- recall, [218](#)
- relative.entropy, [226](#)
- roc.curve, [238](#)
- shannon.entropy, [254](#)
- specificity, [262](#)
- zerooneloss, [379](#)

* Entropy

- cross.entropy, [53](#)
- logloss, [137](#)
- relative.entropy, [226](#)
- shannon.entropy, [254](#)

* Machine learning performance evaluation

- accuracy.cmatrix, [8](#)
- accuracy.factor, [10](#)
- auc.pr.curve.factor, [15](#)

- auc.roc.curve.factor, [21](#)
- baccuracy.cmatrix, [28](#)
- baccuracy.factor, [30](#)
- brier.score.matrix, [36](#)
- ccc.numeric, [39](#)
- ckappa.cmatrix, [44](#)
- ckappa.factor, [46](#)
- cmatrix.factor, [51](#)
- cross.entropy.matrix, [56](#)
- deviance.gamma.numeric, [59](#)
- deviance.poisson.numeric, [63](#)
- deviance.tweedie.numeric, [67](#)
- dor.cmatrix, [71](#)
- dor.factor, [73](#)
- fbeta.cmatrix, [79](#)
- fbeta.factor, [81](#)
- fdr.cmatrix, [87](#)
- fdr.factor, [89](#)
- fer.cmatrix, [94](#)
- fer.factor, [97](#)
- fmi.cmatrix, [102](#)
- fmi.factor, [104](#)
- fpr.cmatrix, [109](#)
- fpr.factor, [112](#)
- gmse.numeric, [117](#)
- hammingloss.cmatrix, [121](#)
- hammingloss.factor, [123](#)
- huberloss.numeric, [127](#)
- jaccard.cmatrix, [132](#)
- jaccard.factor, [134](#)
- logloss.factor, [140](#)
- logloss.integer, [142](#)
- maape.numeric, [146](#)
- mae.numeric, [149](#)
- mape.numeric, [153](#)
- mcc.cmatrix, [157](#)
- mcc.factor, [159](#)
- mpe.numeric, [163](#)
- mse.numeric, [167](#)

- nlr.cmatrix, 171
- nlr.factor, 174
- npv.cmatrix, 179
- npv.factor, 181
- pinball.numeric, 188
- plr.cmatrix, 192
- plr.factor, 195
- pr.curve.factor, 200
- precision.cmatrix, 205
- precision.factor, 208
- rae.numeric, 216
- recall.cmatrix, 221
- recall.factor, 224
- relative.entropy.matrix, 229
- rmse.numeric, 232
- rmsle.numeric, 236
- roc.curve.factor, 240
- rrmse.numeric, 245
- rrse.numeric, 249
- rsq.numeric, 252
- shannon.entropy.matrix, 256
- smape.numeric, 260
- specificity.cmatrix, 265
- specificity.factor, 267
- weighted.accuracy.factor, 270
- weighted.auc.pr.curve.factor, 273
- weighted.auc.roc.curve.factor, 276
- weighted.baccracy.factor, 279
- weighted.brier.score.matrix, 281
- weighted.ccc.numeric, 283
- weighted.ckappa.factor, 285
- weighted.cmatrix.factor, 288
- weighted.deviance.gamma.numeric, 290
- weighted.deviance.poisson.numeric, 292
- weighted.deviance.tweedie.numeric, 294
- weighted.dor.factor, 296
- weighted.fbeta.factor, 299
- weighted.fdr.factor, 302
- weighted.fer.factor, 305
- weighted.fmi.factor, 307
- weighted.fpr.factor, 310
- weighted.gmse.numeric, 313
- weighted.hammingloss.factor, 314
- weighted.huberloss.numeric, 317
- weighted.jaccard.factor, 319
- weighted.logloss.factor, 322
- weighted.logloss.integer, 324
- weighted.maape.numeric, 326
- weighted.mae.numeric, 328
- weighted.mape.numeric, 330
- weighted.mcc.factor, 332
- weighted.mpe.numeric, 334
- weighted.mse.numeric, 336
- weighted.nlr.factor, 338
- weighted.npv.factor, 340
- weighted.pinball.numeric, 343
- weighted.plr.factor, 345
- weighted.pr.curve.factor, 348
- weighted.precision.factor, 351
- weighted.rae.numeric, 354
- weighted.recall.factor, 355
- weighted.rmse.numeric, 358
- weighted.rmsle.numeric, 360
- weighted.roc.curve.factor, 362
- weighted.rrmse.numeric, 365
- weighted.rrse.numeric, 367
- weighted.rsq.numeric, 369
- weighted.smape.numeric, 371
- weighted.specificity.factor, 372
- weighted.zerooneloss.factor, 375
- zerooneloss.cmatrix, 381
- zerooneloss.factor, 384
- * Machine learning**
 - accuracy, 5
 - auc.pr.curve, 12
 - auc.roc.curve, 18
 - baccracy, 25
 - brier.score, 34
 - ccc, 38
 - ckappa, 41
 - cmatrix, 48
 - cross.entropy, 53
 - deviance.gamma, 58
 - deviance.poisson, 61
 - deviance.tweedie, 65
 - dor, 69
 - fbeta, 76
 - fdr, 84
 - fer, 92
 - fmi, 100
 - fpr, 107
 - gmse, 115
 - hammingloss, 118

- huberloss, 125
- jaccard, 129
- logloss, 137
- maape, 144
- mae, 148
- mape, 151
- mcc, 154
- mpe, 162
- mse, 165
- nlr, 169
- npv, 176
- pinball, 186
- plr, 190
- pr.curve, 197
- precision, 203
- rae, 215
- recall, 218
- relative.entropy, 226
- rmse, 231
- rmsle, 234
- roc.curve, 238
- rmse, 243
- rrse, 247
- rsq, 250
- shannon.entropy, 254
- smape, 258
- specificity, 262
- zerooneloss, 379
- * Performance evaluation**
 - accuracy, 5
 - auc.pr.curve, 12
 - auc.roc.curve, 18
 - baccuracy, 25
 - brier.score, 34
 - ccc, 38
 - ckappa, 41
 - cmatrix, 48
 - cross.entropy, 53
 - deviance.gamma, 58
 - deviance.poisson, 61
 - deviance.tweedie, 65
 - dor, 69
 - fbeta, 76
 - fdr, 84
 - fer, 92
 - fmi, 100
 - fpr, 107
 - gmse, 115
 - hammingloss, 118
 - huberloss, 125
 - jaccard, 129
 - logloss, 137
 - maape, 144
 - mae, 148
 - mape, 151
 - mcc, 154
 - mpe, 162
 - mse, 165
 - nlr, 169
 - npv, 176
 - pinball, 186
 - plr, 190
 - pr.curve, 197
 - precision, 203
 - rae, 215
 - recall, 218
 - relative.entropy, 226
 - rmse, 231
 - rmsle, 234
 - roc.curve, 238
 - rmse, 243
 - rrse, 247
 - rsq, 250
 - shannon.entropy, 254
 - smape, 258
 - specificity, 262
 - zerooneloss, 379
- * Proper scoring rules**
 - brier.score, 34
- * Regression**
 - ccc, 38
 - deviance.gamma, 58
 - deviance.poisson, 61
 - deviance.tweedie, 65
 - gmse, 115
 - huberloss, 125
 - maape, 144
 - mae, 148
 - mape, 151
 - mpe, 162
 - mse, 165
 - pinball, 186
 - rae, 215
 - rmse, 231
 - rmsle, 234
 - rmse, 243

- rrse, 247
- rsq, 250
- smape, 258
- * **Statistical learning**
 - accuracy, 5
 - auc.pr.curve, 12
 - auc.roc.curve, 18
 - baccuracy, 25
 - brier.score, 34
 - ccc, 38
 - ckappa, 41
 - cmatrix, 48
 - cross.entropy, 53
 - deviance.gamma, 58
 - deviance.poisson, 61
 - deviance.tweedie, 65
 - dor, 69
 - fbeta, 76
 - fdr, 84
 - fer, 92
 - fmi, 100
 - fpr, 107
 - gmse, 115
 - hammingloss, 118
 - huberloss, 125
 - jaccard, 129
 - logloss, 137
 - maape, 144
 - mae, 148
 - mape, 151
 - mcc, 154
 - mpe, 162
 - mse, 165
 - nlr, 169
 - npv, 176
 - pinball, 186
 - plr, 190
 - pr.curve, 197
 - precision, 203
 - rae, 215
 - recall, 218
 - relative.entropy, 226
 - rmse, 231
 - rmsle, 234
 - roc.curve, 238
 - rrmse, 243
 - rrse, 247
 - rsq, 250
 - shannon.entropy, 254
 - smape, 258
 - specificity, 262
 - zerooneloss, 379
- * **Supervised Learning**
 - accuracy, 5
 - auc.pr.curve, 12
 - auc.roc.curve, 18
 - baccuracy, 25
 - brier.score, 34
 - ccc, 38
 - ckappa, 41
 - cmatrix, 48
 - cross.entropy, 53
 - deviance.gamma, 58
 - deviance.poisson, 61
 - deviance.tweedie, 65
 - dor, 69
 - fbeta, 76
 - fdr, 84
 - fer, 92
 - fmi, 100
 - fpr, 107
 - gmse, 115
 - hammingloss, 118
 - huberloss, 125
 - jaccard, 129
 - logloss, 137
 - maape, 144
 - mae, 148
 - mape, 151
 - mcc, 154
 - mpe, 162
 - mse, 165
 - nlr, 169
 - npv, 176
 - pinball, 186
 - plr, 190
 - pr.curve, 197
 - precision, 203
 - rae, 215
 - recall, 218
 - relative.entropy, 226
 - rmse, 231
 - rmsle, 234
 - roc.curve, 238
 - rrmse, 243
 - rrse, 247

- rsq, 250
- shannon.entropy, 254
- smape, 258
- specificity, 262
- zerooneloss, 379
- * **Tools**
 - auc.xy, 24
- * **Utilities**
 - preorder, 211
 - presort, 213
- * **classification**
 - accuracy, 5
 - accuracy.cmatrix, 8
 - accuracy.factor, 10
 - auc.pr.curve, 12
 - auc.pr.curve.factor, 15
 - auc.roc.curve, 18
 - auc.roc.curve.factor, 21
 - baccuracy, 25
 - baccuracy.cmatrix, 28
 - baccuracy.factor, 30
 - brier.score, 34
 - brier.score.matrix, 36
 - ccc, 38
 - ckappa, 41
 - ckappa.cmatrix, 44
 - ckappa.factor, 46
 - cmatrix, 48
 - cmatrix.factor, 51
 - cross.entropy, 53
 - cross.entropy.matrix, 56
 - deviance.gamma, 58
 - deviance.poisson, 61
 - deviance.tweedie, 65
 - dor, 69
 - dor.cmatrix, 71
 - dor.factor, 73
 - fbeta, 76
 - fbeta.cmatrix, 79
 - fbeta.factor, 81
 - fdr, 84
 - fdr.cmatrix, 87
 - fdr.factor, 89
 - fer, 92
 - fer.cmatrix, 94
 - fer.factor, 97
 - fmi, 100
 - fmi.cmatrix, 102
 - fmi.factor, 104
 - fpr, 107
 - fpr.cmatrix, 109
 - fpr.factor, 112
 - gmse, 115
 - hammingloss, 118
 - hammingloss.cmatrix, 121
 - hammingloss.factor, 123
 - huberloss, 125
 - jaccard, 129
 - jaccard.cmatrix, 132
 - jaccard.factor, 134
 - logloss, 137
 - logloss.factor, 140
 - logloss.integer, 142
 - maape, 144
 - mae, 148
 - mape, 151
 - mcc, 154
 - mcc.cmatrix, 157
 - mcc.factor, 159
 - mpe, 162
 - mse, 165
 - nlr, 169
 - nlr.cmatrix, 171
 - nlr.factor, 174
 - npv, 176
 - npv.cmatrix, 179
 - npv.factor, 181
 - pinball, 186
 - plr, 190
 - plr.cmatrix, 192
 - plr.factor, 195
 - pr.curve, 197
 - pr.curve.factor, 200
 - precision, 203
 - precision.cmatrix, 205
 - precision.factor, 208
 - rae, 215
 - recall, 218
 - recall.cmatrix, 221
 - recall.factor, 224
 - relative.entropy, 226
 - relative.entropy.matrix, 229
 - rmse, 231
 - rmsle, 234
 - roc.curve, 238
 - roc.curve.factor, 240

- rmse, 243
- rrse, 247
- rsq, 250
- shannon.entropy, 254
- shannon.entropy.matrix, 256
- smape, 258
- specificity, 262
- specificity.cmatrix, 265
- specificity.factor, 267
- weighted.accuracy.factor, 270
- weighted.auc.pr.curve.factor, 273
- weighted.auc.roc.curve.factor, 276
- weighted.baccracy.factor, 279
- weighted.brier.score.matrix, 281
- weighted.ckappa.factor, 285
- weighted.cmatrix.factor, 288
- weighted.dor.factor, 296
- weighted.fbeta.factor, 299
- weighted.fdr.factor, 302
- weighted.fer.factor, 305
- weighted.fmi.factor, 307
- weighted.fpr.factor, 310
- weighted.hammingloss.factor, 314
- weighted.jaccard.factor, 319
- weighted.logloss.factor, 322
- weighted.logloss.integer, 324
- weighted.mcc.factor, 332
- weighted.nlr.factor, 338
- weighted.npv.factor, 340
- weighted.plr.factor, 345
- weighted.pr.curve.factor, 348
- weighted.precision.factor, 351
- weighted.recall.factor, 355
- weighted.roc.curve.factor, 362
- weighted.specificity.factor, 372
- weighted.zeroone loss.factor, 375
- zeroone loss, 379
- zeroone loss.cmatrix, 381
- zeroone loss.factor, 384
- * datasets**
 - banknote, 33
 - obesity, 184
 - wine.quality, 378
- * entropy**
 - cross.entropy, 53
 - cross.entropy.matrix, 56
 - logloss, 137
 - logloss.factor, 140
 - logloss.integer, 142
 - relative.entropy, 226
 - relative.entropy.matrix, 229
 - shannon.entropy, 254
 - shannon.entropy.matrix, 256
 - weighted.logloss.factor, 322
 - weighted.logloss.integer, 324
- * evaluation**
 - accuracy, 5
 - accuracy.cmatrix, 8
 - accuracy.factor, 10
 - auc.pr.curve, 12
 - auc.pr.curve.factor, 15
 - auc.roc.curve, 18
 - auc.roc.curve.factor, 21
 - baccracy, 25
 - baccracy.cmatrix, 28
 - baccracy.factor, 30
 - brier.score, 34
 - brier.score.matrix, 36
 - ccc, 38
 - ccc.numeric, 39
 - ckappa, 41
 - ckappa.cmatrix, 44
 - ckappa.factor, 46
 - cmatrix, 48
 - cmatrix.factor, 51
 - cross.entropy, 53
 - cross.entropy.matrix, 56
 - deviance.gamma, 58
 - deviance.gamma.numeric, 59
 - deviance.poisson, 61
 - deviance.poisson.numeric, 63
 - deviance.tweedie, 65
 - deviance.tweedie.numeric, 67
 - dor, 69
 - dor.cmatrix, 71
 - dor.factor, 73
 - fbeta, 76
 - fbeta.cmatrix, 79
 - fbeta.factor, 81
 - fdr, 84
 - fdr.cmatrix, 87
 - fdr.factor, 89
 - fer, 92
 - fer.cmatrix, 94
 - fer.factor, 97
 - fmi, 100

- fmi.cmatrix, 102
- fmi.factor, 104
- fpr, 107
- fpr.cmatrix, 109
- fpr.factor, 112
- gmse, 115
- gmse.numeric, 117
- hammingloss, 118
- hammingloss.cmatrix, 121
- hammingloss.factor, 123
- huberloss, 125
- huberloss.numeric, 127
- jaccard, 129
- jaccard.cmatrix, 132
- jaccard.factor, 134
- logloss, 137
- logloss.factor, 140
- logloss.integer, 142
- maape, 144
- maape.numeric, 146
- mae, 148
- mae.numeric, 149
- mape, 151
- mape.numeric, 153
- mcc, 154
- mcc.cmatrix, 157
- mcc.factor, 159
- mpe, 162
- mpe.numeric, 163
- mse, 165
- mse.numeric, 167
- nlr, 169
- nlr.cmatrix, 171
- nlr.factor, 174
- npv, 176
- npv.cmatrix, 179
- npv.factor, 181
- pinball, 186
- pinball.numeric, 188
- plr, 190
- plr.cmatrix, 192
- plr.factor, 195
- pr.curve, 197
- pr.curve.factor, 200
- precision, 203
- precision.cmatrix, 205
- precision.factor, 208
- rae, 215
- rae.numeric, 216
- recall, 218
- recall.cmatrix, 221
- recall.factor, 224
- relative.entropy, 226
- relative.entropy.matrix, 229
- rmse, 231
- rmse.numeric, 232
- rmsle, 234
- rmsle.numeric, 236
- roc.curve, 238
- roc.curve.factor, 240
- rrmse, 243
- rrmse.numeric, 245
- rrse, 247
- rrse.numeric, 249
- rsq, 250
- rsq.numeric, 252
- shannon.entropy, 254
- shannon.entropy.matrix, 256
- smape, 258
- smape.numeric, 260
- specificity, 262
- specificity.cmatrix, 265
- specificity.factor, 267
- weighted.accuracy.factor, 270
- weighted.auc.pr.curve.factor, 273
- weighted.auc.roc.curve.factor, 276
- weighted.baccuracy.factor, 279
- weighted.brier.score.matrix, 281
- weighted.ccc.numeric, 283
- weighted.ckappa.factor, 285
- weighted.cmatrix.factor, 288
- weighted.deviance.gamma.numeric, 290
- weighted.deviance.poisson.numeric, 292
- weighted.deviance.tweedie.numeric, 294
- weighted.dor.factor, 296
- weighted.fbeta.factor, 299
- weighted.fdr.factor, 302
- weighted.fer.factor, 305
- weighted.fmi.factor, 307
- weighted.fpr.factor, 310
- weighted.gmse.numeric, 313
- weighted.hammingloss.factor, 314
- weighted.huberloss.numeric, 317

- weighted.jaccard.factor, 319
- weighted.logloss.factor, 322
- weighted.logloss.integer, 324
- weighted.maape.numeric, 326
- weighted.mae.numeric, 328
- weighted.mape.numeric, 330
- weighted.mcc.factor, 332
- weighted.mpe.numeric, 334
- weighted.mse.numeric, 336
- weighted.nlr.factor, 338
- weighted.npv.factor, 340
- weighted.pinball.numeric, 343
- weighted.plr.factor, 345
- weighted.pr.curve.factor, 348
- weighted.precision.factor, 351
- weighted.rae.numeric, 354
- weighted.recall.factor, 355
- weighted.rmse.numeric, 358
- weighted.rmsle.numeric, 360
- weighted.roc.curve.factor, 362
- weighted.rrmse.numeric, 365
- weighted.rrse.numeric, 367
- weighted.rsq.numeric, 369
- weighted.smape.numeric, 371
- weighted.specificity.factor, 372
- weighted.zerooneloss.factor, 375
- zerooneloss, 379
- zerooneloss.cmatrix, 381
- zerooneloss.factor, 384
- * **regression**
 - accuracy, 5
 - auc.pr.curve, 12
 - auc.roc.curve, 18
 - baccuracy, 25
 - brier.score, 34
 - ccc, 38
 - ccc.numeric, 39
 - ckappa, 41
 - cmatrix, 48
 - cross.entropy, 53
 - deviance.gamma, 58
 - deviance.gamma.numeric, 59
 - deviance.poisson, 61
 - deviance.poisson.numeric, 63
 - deviance.tweedie, 65
 - deviance.tweedie.numeric, 67
 - dor, 69
 - fbeta, 76
 - fdr, 84
 - fer, 92
 - fmi, 100
 - fpr, 107
 - gmse, 115
 - gmse.numeric, 117
 - hammingloss, 118
 - huberloss, 125
 - huberloss.numeric, 127
 - jaccard, 129
 - logloss, 137
 - maape, 144
 - maape.numeric, 146
 - mae, 148
 - mae.numeric, 149
 - mape, 151
 - mape.numeric, 153
 - mcc, 154
 - mpe, 162
 - mpe.numeric, 163
 - mse, 165
 - mse.numeric, 167
 - nlr, 169
 - npv, 176
 - pinball, 186
 - pinball.numeric, 188
 - plr, 190
 - pr.curve, 197
 - precision, 203
 - rae, 215
 - rae.numeric, 216
 - recall, 218
 - relative.entropy, 226
 - rmse, 231
 - rmse.numeric, 232
 - rmsle, 234
 - rmsle.numeric, 236
 - roc.curve, 238
 - rrmse, 243
 - rrmse.numeric, 245
 - rrse, 247
 - rrse.numeric, 249
 - rsq, 250
 - rsq.numeric, 252
 - shannon.entropy, 254
 - smape, 258
 - smape.numeric, 260
 - specificity, 262

- weighted.ccc.numeric, 283
 - weighted.deviance.gamma.numeric, 290
 - weighted.deviance.poisson.numeric, 292
 - weighted.deviance.tweedie.numeric, 294
 - weighted.gmse.numeric, 313
 - weighted.huberloss.numeric, 317
 - weighted.maafe.numeric, 326
 - weighted.mae.numeric, 328
 - weighted.mape.numeric, 330
 - weighted.mpe.numeric, 334
 - weighted.mse.numeric, 336
 - weighted.pinball.numeric, 343
 - weighted.rae.numeric, 354
 - weighted.rmse.numeric, 358
 - weighted.rmsle.numeric, 360
 - weighted.rrmse.numeric, 365
 - weighted.rrse.numeric, 367
 - weighted.rsq.numeric, 369
 - weighted.smape.numeric, 371
 - zerooneloss, 379
- accuracy, 5, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 187, 189, 191, 194, 196, 199, 202, 205, 207, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 269, 275, 278, 280, 283, 285, 287, 289, 290, 292, 293, 296, 298, 301, 304, 306, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 339, 340, 342, 345, 347, 350, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- accuracy(), 5, 8, 10, 270
- accuracy.cmatrix, 6, 8
- accuracy.cmatrix(), 6, 8, 11, 271
- accuracy.factor, 6, 10
- accuracy.factor(), 6, 8, 11, 271
- auc.pr.curve, 7, 9, 11, 12, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 187, 189, 191, 194, 196, 197, 199, 200, 202, 205, 207, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 269, 272, 278, 280, 283, 285, 287, 289, 290, 292, 293, 296, 298, 301, 304, 306, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 339, 340, 342, 345, 347, 348, 350, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- auc.pr.curve(), 12, 15, 273
- auc.pr.curve.factor, 13, 15
- auc.roc.curve, 7, 9, 11, 14, 17, 18, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 187, 189, 191, 194, 196, 199, 202, 205, 207, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237–239, 241, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 269, 272, 275, 280, 283, 285, 287, 289, 290, 292, 293, 296, 298, 301, 304, 306, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 339, 340, 342,

- 345, 347, 350, 353, 355, 357, 358,
 360, 361, 363, 364, 366, 368, 370,
 372, 374, 375, 377, 381, 383, 385
 auc.roc.curve(), 18, 21, 276
 auc.roc.curve.factor, 19, 21
 auc.xy, 24
 auc.xy(), 24
 auc.xy.numeric, 24, 25
 baccaccuracy, 7, 9, 11, 14, 17, 20, 23, 25, 35, 37,
 39, 41, 43, 45, 48, 50, 53, 55, 57, 59,
 61, 62, 64, 66, 68, 70, 73, 75, 78, 80,
 83, 86, 88, 91, 94, 96, 99, 101, 103,
 104, 106, 109, 111, 114, 116, 118,
 120, 122, 124, 125, 127, 128, 131,
 134, 136, 139, 141, 143, 145, 147,
 149, 151, 152, 154, 156, 158, 159,
 161, 163, 165, 166, 168, 170, 173,
 175, 178, 180, 183, 187, 189, 191,
 194, 196, 199, 202, 205, 207, 210,
 216, 218, 220, 223, 226, 228, 230,
 232, 234, 235, 237, 239, 242, 245,
 246, 248, 250, 252, 253, 255, 257,
 259, 261, 264, 267, 269, 272, 275,
 278, 283, 285, 287, 289, 290, 292,
 293, 296, 298, 301, 304, 306, 309,
 312, 314, 316, 318, 321, 323, 325,
 328, 329, 331, 333, 336, 337, 339,
 340, 342, 345, 347, 350, 353, 355,
 357, 358, 360, 361, 364, 366, 368,
 370, 372, 374, 375, 377, 381, 383,
 385
 baccaccuracy(), 25, 28, 30, 31, 279
 baccaccuracy.cmatrix, 26, 28
 baccaccuracy.cmatrix(), 26, 29, 31, 280
 baccaccuracy.factor, 26, 30
 baccaccuracy.factor(), 26, 29, 31, 280
 banknote, 33
 brier.score, 7, 9, 11, 14, 17, 20, 23, 27, 29,
 30, 32, 34, 39, 41, 43, 45, 48, 50, 53,
 55, 57, 59, 61, 62, 64, 66, 68, 70, 73,
 75, 78, 80, 83, 86, 88, 91, 94, 96, 99,
 101, 103, 104, 106, 109, 111, 114,
 116, 118, 120, 122, 124, 125, 127,
 128, 131, 134, 136, 139, 141, 143,
 145, 147, 149, 151, 152, 154, 156,
 158, 159, 161, 163, 165, 166, 168,
 170, 173, 175, 178, 180, 183, 187,
 189, 191, 194, 196, 199, 202, 205,
 207, 210, 216, 218, 220, 223, 226,
 228, 230, 232, 234, 235, 237, 239,
 242, 245, 246, 248, 250, 252, 253,
 255, 257, 259, 261, 264, 267, 269,
 272, 275, 278, 280, 285, 287, 289,
 290, 292, 293, 296, 298, 301, 304,
 306, 309, 312, 314, 316, 318, 321,
 323, 325, 328, 329, 331, 333, 336,
 337, 339, 340, 342, 345, 347, 350,
 353, 355, 357, 358, 360, 361, 364,
 366, 368, 370, 372, 374, 375, 377,
 381, 383, 385
 brier.score(), 34, 36, 281, 282
 brier.score.matrix, 34, 36
 ccc, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37,
 38, 43, 45, 48, 50, 53, 55, 57, 59, 61,
 62, 64, 66, 68, 70, 73, 75, 78, 80, 83,
 86, 88, 91, 94, 96, 99, 101, 104, 106,
 109, 111, 114, 116, 118, 120, 122,
 125, 127, 128, 131, 134, 136, 139,
 141, 143, 145, 147, 149–152, 154,
 156, 159, 161, 163, 165, 166, 168,
 170, 173, 175, 178, 180, 183, 187,
 189, 191, 194, 196, 199, 202, 205,
 207, 210, 216–218, 220, 223, 226,
 228, 230, 232, 234, 235, 237, 239,
 242, 245, 246, 248, 250, 252, 253,
 255, 257, 259, 261, 264, 267, 269,
 272, 275, 278, 280, 283, 287, 290,
 292, 293, 296, 298, 301, 304, 306,
 309, 312, 314, 316, 318, 321, 323,
 325, 328, 329, 331, 333, 336, 337,
 340, 342, 345, 347, 350, 353, 355,
 358, 360, 361, 364, 366, 368, 370,
 372, 375, 377, 381, 383, 385
 ccc(), 38, 40, 284
 ccc.numeric, 38, 39
 character, 198, 201, 239, 242, 349, 363
 ckappa, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32,
 35, 37, 39, 41, 41, 50, 53, 55, 57, 59,
 61, 62, 64, 66, 68, 70, 73, 75, 78, 80,
 83, 86, 88, 91, 94, 96, 99, 101, 103,
 104, 106, 109, 111, 114, 116, 118,
 120, 122, 124, 125, 127, 128, 131,
 134, 136, 139, 141, 143, 145, 147,
 149, 151, 152, 154, 156, 158, 159,
 161, 163, 165, 166, 168, 170, 173,
 175, 178, 180, 183, 187, 189, 191,

- 194, 196, 199, 202, 205, 207, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 269, 272, 275, 278, 280, 283, 285, 289, 290, 292, 293, 296, 298, 301, 304, 306, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 339, 340, 342, 345, 347, 350, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- ckappa(), 41, 44, 46, 285, 286
- ckappa.cmatrix, 42, 44
- ckappa.cmatrix(), 42, 44, 47, 286
- ckappa.factor, 42, 46
- ckappa.factor(), 42, 44, 47, 286
- cmatrix, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 48, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 187, 189, 191, 194, 196, 199, 202, 205, 207, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 269, 272, 275, 278, 280, 283, 285, 287, 292, 293, 296, 298, 301, 304, 306, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 339, 340, 342, 345, 347, 350, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- cmatrix(), 6, 8–11, 26–29, 31, 42, 44, 45, 47–49, 51, 69, 70, 72, 74, 76, 77, 79, 80, 82, 84, 85, 87, 88, 90, 92, 93, 95, 98, 100, 101, 103, 105, 107, 108, 110, 111, 113, 119–122, 124, 129, 130, 132, 133, 135, 155–158, 160, 169, 170, 172, 174, 176, 177, 179, 180, 182, 190, 191, 193, 195, 203, 204, 206, 207, 209, 219, 221, 222, 224, 262, 263, 265, 266, 268, 271, 279, 280, 286, 288, 297, 299, 300, 302, 303, 305, 308, 310, 311, 315, 319, 320, 332, 333, 339, 341, 346, 351, 352, 356, 373, 376, 379, 380, 382, 384
- cmatrix.factor, 49, 51
- cross.entropy, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 53, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 187, 189, 191, 194, 196, 199, 202, 205, 207, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257–259, 261, 264, 267, 269, 272, 275, 278, 280, 283, 285, 287, 289, 290, 292, 293, 296, 298, 301, 304, 306, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 339, 340, 342, 345, 347, 350, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- cross.entropy(), 53, 54, 56
- cross.entropy.matrix, 54, 56
- csi (jaccard), 129
- csi(), 130, 133, 136, 320
- data.frame, 13, 16, 19, 21, 198, 201, 239, 242, 273, 276, 349, 363
- deviance.gamma, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 58, 62, 64, 66, 68, 70, 73, 75, 78, 80, 83, 86, 88, 91, 94, 96, 99, 101, 104, 106, 109, 111, 114, 116, 118, 120, 122, 125, 127, 128, 131, 134, 136, 139, 141, 143, 145, 147, 149–152, 154, 156, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 187, 189, 191, 194, 196,

- 199, 202, 205, 207, 210, 216–218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 269, 272, 275, 278, 280, 283, 285, 287, 290, 293, 296, 298, 301, 304, 306, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 340, 342, 345, 347, 350, 353, 355, 358, 360, 361, 364, 366, 368, 370, 372, 375, 377, 381, 383, 385
- deviance.gamma(), 58, 60, 66, 68, 291, 295
- deviance.gamma.numeric, 58, 59
- deviance.poisson, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 61, 66, 68, 70, 73, 75, 78, 80, 83, 86, 88, 91, 94, 96, 99, 101, 104, 106, 109, 111, 114, 116, 118, 120, 122, 125, 127, 128, 131, 134, 136, 139, 141, 143, 145, 147, 149–152, 154, 156, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 187, 189, 191, 194, 196, 199, 202, 205, 207, 210, 216–218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 269, 272, 275, 278, 280, 283, 285, 287, 290, 292, 296, 298, 301, 304, 306, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 340, 342, 345, 347, 350, 353, 355, 358, 360, 361, 364, 366, 368, 370, 372, 375, 377, 381, 383, 385
- deviance.poisson(), 61, 63, 66, 68, 292, 293, 295
- deviance.poisson.numeric, 62, 63
- deviance.tweedie, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 65, 70, 73, 75, 78, 80, 83, 86, 88, 91, 94, 96, 99, 101, 104, 106, 109, 111, 114, 116, 118, 120, 122, 125, 127, 128, 131, 134, 137, 139, 141, 143, 145, 147, 149–152, 154, 156, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 187–189, 192, 194, 196, 199, 202, 205, 207, 210, 216–218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 270, 272, 275, 278, 280, 283, 285, 287, 293, 298, 301, 304, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 340, 342, 345, 347, 350, 353, 355, 358, 360, 361, 364, 366, 368, 370, 372, 375, 377, 381, 383, 385
- deviance.tweedie(), 65, 67, 294
- deviance.tweedie.numeric, 65, 67
- dor, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 69, 78, 80, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 137, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 188, 189, 191, 192, 194, 196, 199, 202, 205, 207, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 269, 270, 272, 275, 278, 280, 283, 285, 287, 289, 290, 292, 294, 296, 298, 301, 304, 306, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 339, 340, 342, 345, 347, 350, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- dor(), 69, 71, 74
- dor.cmatrix, 70, 71
- dor.cmatrix(), 70, 72, 74
- dor.factor, 70, 73
- dor.factor(), 70, 72, 74
- double, 6, 9, 11, 13, 14, 16, 17, 19, 20, 22, 24, 25, 27, 29, 32, 34–36, 38–40, 42, 43, 45, 47, 50, 54, 56–60, 62, 64–68, 70, 72, 75, 77, 80, 82, 83, 85, 88, 90, 91, 93, 95, 96, 98, 101, 103, 105, 108,

- 111, 113, 114, 116, 117, 120, 122, 124, 126, 128, 130, 133, 136, 138, 140, 141, 143, 145, 147, 148, 150, 152, 153, 156, 158, 160, 162–164, 166, 168, 170, 172, 175, 177, 180, 182, 183, 187, 189, 191, 193, 196, 198, 201, 204, 207, 209, 210, 215, 217, 219, 220, 222, 225, 227, 229–233, 235, 237, 239, 242, 244, 246, 248, 249, 251, 253, 255, 257, 259, 261, 263, 266, 269, 271, 274, 277, 280, 282, 284, 286, 287, 289, 291, 293, 295, 297, 300, 303, 306, 308, 309, 311, 313, 316, 318, 320, 322, 323, 325, 327, 329, 331, 333, 335, 337, 339, 342, 344, 346, 349, 352, 354, 357, 359, 361, 363, 366, 368–371, 374, 376, 377, 380, 382, 385
- factor, 6, 11, 13, 16, 19, 22, 26, 31, 42, 47, 50, 52, 70, 75, 77, 82, 85, 90, 93, 98, 101, 105, 108, 113, 120, 124, 130, 136, 138, 140, 143, 156, 160, 170, 175, 177, 182, 191, 196, 198, 201, 204, 209, 219, 225, 239, 241, 242, 263, 269, 271, 274, 277, 280, 286, 289, 297, 300, 303, 306, 308, 311, 316, 320, 322, 325, 333, 339, 342, 346, 349, 352, 357, 363, 374, 376, 380, 385
- fallout (fpr), 107
- fallout(), 108, 111, 114, 311
- FALSE, 24–26, 29, 31, 38, 40, 187, 189, 280, 284, 344
- fbeta, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 76, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 137, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 188, 189, 191, 192, 194, 196, 199, 202, 205, 207, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 269, 270, 272, 275, 278, 280, 283, 285, 287, 289, 290, 292, 294, 296, 298, 301, 306, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 339, 340, 342, 345, 347, 350, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- fbeta(), 76, 79, 81, 82, 299
- fbeta.cmatrix, 77, 79
- fbeta.cmatrix(), 77, 79, 82, 300
- fbeta.factor, 77, 81
- fbeta.factor(), 77, 79, 82, 300
- fdr, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 83, 84, 94, 96, 99, 101, 103, 104, 106, 109, 111, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 137, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 183, 188, 189, 191, 192, 194, 196, 199, 202, 205, 207, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 242, 245, 246, 248, 250, 252, 253, 255, 257, 259, 261, 264, 267, 269, 270, 272, 275, 278, 280, 283, 285, 287, 289, 290, 292, 294, 296, 298, 301, 306, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 336, 337, 339, 340, 342, 345, 347, 350, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- fdr(), 84, 87, 89, 302
- fdr.cmatrix, 85, 87
- fdr.cmatrix(), 85, 87, 90, 303
- fdr.factor, 85, 89
- fdr.factor(), 85, 87, 90, 303
- fer, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 83, 86, 88, 91, 92, 101, 103, 104, 106, 109, 111, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134,

- 136, 137, 139, 141, 143, 145, 147,
 149, 151, 152, 154, 156, 158, 159,
 161, 163, 165, 166, 168, 170, 173,
 175, 178, 180, 183, 188, 189, 191,
 192, 194, 196, 199, 202, 205, 207,
 210, 216, 218, 220, 223, 226, 228,
 230, 232, 234, 235, 237, 239, 242,
 245, 246, 248, 250, 252, 253, 255,
 257, 259, 261, 264, 267, 269, 270,
 272, 275, 278, 280, 283, 285, 287,
 289, 290, 292, 294, 296, 298, 301,
 304, 309, 312, 314, 316, 318, 321,
 323, 325, 328, 329, 331, 333, 336,
 337, 339, 340, 342, 345, 347, 350,
 353, 355, 357, 358, 360, 361, 364,
 366, 368, 370, 372, 374, 375, 377,
 381, 383, 385
 fer(), 92, 94, 95, 97, 305
 fer.cmatrix, 93, 94
 fer.cmatrix(), 93, 95, 98, 305
 fer.factor, 93, 97
 fer.factor(), 93, 95, 98, 305
 fmi, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35,
 37, 39, 41, 43, 45, 48, 50, 53, 55, 57,
 59, 61, 62, 64, 66, 68, 70, 73, 75, 78,
 80, 83, 86, 88, 91, 94, 96, 99, 100,
 109, 111, 114, 116, 118, 120, 122,
 124, 125, 127, 128, 131, 134, 136,
 137, 139, 141, 143, 145, 147, 149,
 151, 152, 154, 156, 158, 159, 161,
 163, 165, 166, 168, 170, 173, 175,
 178, 180, 183, 188, 189, 191, 192,
 194, 196, 199, 202, 205, 207, 210,
 216, 218, 220, 223, 226, 228, 230,
 232, 234, 235, 237, 239, 242, 245,
 246, 248, 250, 252, 253, 255, 257,
 259, 261, 264, 267, 269, 270, 272,
 275, 278, 280, 283, 285, 287, 289,
 290, 292, 294, 296, 298, 301, 304,
 306, 307, 312, 314, 316, 318, 321,
 323, 325, 328, 329, 331, 333, 336,
 337, 339, 340, 342, 345, 347, 350,
 353, 355, 357, 358, 360, 361, 364,
 366, 368, 370, 372, 374, 375, 377,
 381, 383, 385
 fmi(), 100, 102, 104, 105, 307, 308
 fmi.cmatrix, 101, 102
 fmi.cmatrix(), 100, 103, 105, 308
 fmi.factor, 101, 104
 fmi.factor(), 100, 103, 105, 308
 fpr, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35,
 37, 39, 41, 43, 45, 48, 50, 53, 55, 57,
 59, 61, 62, 64, 66, 68, 70, 73, 75, 78,
 80, 83, 86, 88, 91, 94, 96, 99, 101,
 103, 104, 106, 107, 116, 118, 120,
 122, 124, 125, 127, 128, 131, 134,
 136, 137, 139, 141, 143, 145, 147,
 149, 151, 152, 154, 156, 158, 159,
 161, 163, 165, 166, 168, 170, 173,
 175, 178, 180, 183, 188, 189, 191,
 192, 194, 196, 199, 202, 205, 207,
 210, 216, 218, 220, 223, 226, 228,
 230, 232, 234, 235, 237, 239, 242,
 245, 246, 248, 250, 252, 253, 255,
 257, 259, 261, 264, 267, 269, 270,
 272, 275, 278, 280, 283, 285, 287,
 289, 290, 292, 294, 296, 298, 301,
 304, 306, 307, 309, 314, 316, 318,
 321, 323, 325, 328, 329, 331, 333,
 336, 337, 339, 340, 342, 345, 347,
 350, 353, 355, 357, 358, 360, 361,
 364, 366, 368, 370, 372, 374, 375,
 377, 381, 383, 385
 fpr(), 107, 110, 112, 113, 239, 242, 310, 363
 fpr.cmatrix, 108, 109
 fpr.cmatrix(), 107, 110, 113, 311
 fpr.factor, 108, 112
 fpr.factor(), 107, 110, 113, 311
 gmse, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35,
 37, 39, 41, 43, 45, 48, 50, 53, 55, 57,
 59, 61, 62, 64, 66, 68, 70, 73, 75, 78,
 80, 83, 86, 88, 91, 94, 96, 99, 101,
 104, 106, 109, 111, 114, 115, 120,
 122, 125, 127, 128, 131, 134, 137,
 139, 141, 143, 145, 147, 149–152,
 154, 156, 159, 161, 163, 165, 166,
 168, 170, 173, 175, 178, 180, 183,
 187–189, 192, 194, 196, 199, 202,
 205, 207, 210, 216–218, 220, 223,
 226, 228, 230, 232, 234, 235, 237,
 239, 242, 245, 246, 248, 250, 252,
 253, 255, 257, 259, 261, 264, 267,
 270, 272, 275, 278, 280, 283, 285,
 287, 290, 292–294, 296, 298, 301,
 304, 307, 309, 312, 316, 318, 321,
 323, 325, 328, 329, 331, 333, 336,

- 337, 340, 342, 345, 347, 350, 353,
355, 358, 360, 361, 364, 366, 368,
370, 372, 375, 377, 381, 383, 385
- gmse(), 115, 117, 313
- gmse.numeric, 116, 117
- hammingloss, 7, 9, 11, 14, 17, 20, 23, 27, 29,
30, 32, 35, 37, 39, 41, 43, 45, 48, 50,
53, 55, 57, 59, 61, 62, 64, 66, 68, 70,
73, 75, 78, 80, 83, 86, 88, 91, 94, 96,
99, 101, 103, 104, 106, 109, 111,
114, 116, 118, 118, 127, 128, 131,
134, 136, 137, 139, 141, 143, 145,
147, 149, 151, 152, 154, 156, 158,
159, 161, 163, 165, 166, 168, 170,
173, 175, 178, 180, 183, 188, 189,
191, 192, 194, 196, 199, 202, 205,
207, 210, 216, 218, 220, 223, 226,
228, 230, 232, 234, 235, 237, 239,
242, 245, 246, 248, 250, 252, 253,
255, 257, 259, 261, 264, 267, 269,
270, 272, 275, 278, 280, 283, 285,
287, 289, 290, 292, 294, 296, 298,
301, 304, 306, 307, 309, 312, 314,
318, 321, 323, 325, 328, 329, 331,
333, 336, 337, 339, 340, 342, 345,
347, 350, 353, 355, 357, 358, 360,
361, 364, 366, 368, 370, 372, 374,
375, 377, 381, 383, 385
- hammingloss(), 118, 119, 121, 123, 315
- hammingloss.cmatrix, 119, 121
- hammingloss.cmatrix(), 119, 122, 124, 315
- hammingloss.factor, 119, 123
- hammingloss.factor(), 119, 122, 124, 315
- huberloss, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32,
35, 37, 39, 41, 43, 45, 48, 50, 53, 55,
57, 59, 61, 62, 64, 66, 68, 70, 73, 75,
78, 80, 83, 86, 88, 91, 94, 96, 99,
101, 104, 106, 109, 111, 114, 116,
118, 120, 122, 125, 125, 131, 134,
137, 139, 141, 143, 145, 147,
149–152, 154, 156, 159, 161, 163,
165, 166, 168, 170, 173, 175, 178,
180, 183, 187–189, 192, 194, 196,
199, 202, 205, 208, 210, 216–218,
220, 223, 226, 228, 230, 232, 234,
235, 237, 240, 242, 245, 246, 248,
250, 252, 253, 255, 257, 259, 261,
264, 267, 270, 272, 275, 278, 280,
283, 285, 287, 290, 292–294, 296,
298, 301, 304, 307, 309, 312, 314,
316, 321, 323, 325, 328, 329, 331,
333, 336, 337, 340, 342, 345, 347,
350, 353, 355, 358, 360, 361, 364,
366, 368, 370, 372, 375, 377, 381,
383, 385
- huberloss(), 125–127, 317
- huberloss.numeric, 126, 127
- integer, 6, 11, 13, 14, 16, 19, 22, 24–26, 31,
42, 47, 50, 52, 54, 56, 70, 75, 77, 80,
82, 85, 88, 90, 93, 95, 98, 101, 105,
108, 111, 113, 120, 124, 130, 133,
136, 138, 140, 143, 156, 160, 170,
175, 177, 180, 182, 186, 191, 196,
198, 201, 204, 207, 209, 219, 222,
225, 227, 229, 239, 241, 242, 255,
257, 263, 266, 269, 271, 274, 277,
280, 286, 289, 297, 300, 303, 306,
308, 311, 316, 320, 322, 325, 333,
339, 342, 346, 349, 352, 357, 363,
374, 376, 380, 385
- IQR, 244, 246, 366
- jaccard, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30,
32, 35, 37, 39, 41, 43, 45, 48, 50, 53,
55, 57, 59, 61, 62, 64, 66, 68, 70, 73,
75, 78, 80, 81, 83, 86, 88, 91, 94, 96,
99, 101, 103, 104, 106, 109, 111,
112, 114, 116, 118, 120, 122, 124,
125, 127, 128, 129, 139, 141, 143,
145, 147, 149, 151, 152, 154, 156,
158, 159, 161, 163, 165, 166, 168,
170, 173, 175, 178, 180, 183, 188,
189, 191, 192, 194, 196, 199, 202,
205, 207, 208, 210, 216, 218, 220,
223, 226, 228, 230, 232, 234, 235,
237, 239, 240, 242, 245, 246, 248,
250, 252, 253, 255, 257, 259, 261,
264, 267, 269, 270, 272, 275, 278,
280, 283, 285, 287, 289, 290, 292,
294, 296, 298, 301, 304, 306, 307,
309, 312, 314, 316, 318, 323, 325,
328, 329, 331, 333, 336, 337, 340,
342, 345, 347, 350, 353, 355, 357,
358, 360, 361, 364, 366, 368, 370,
372, 374, 375, 377, 381, 383, 385
- jaccard(), 129, 132, 135, 319

- jaccard.cmatrix, [130](#), [132](#)
- jaccard.cmatrix(), [130](#), [132](#), [135](#), [320](#)
- jaccard.factor, [130](#), [134](#)
- jaccard.factor(), [130](#), [132](#), [135](#), [320](#)
- length, [5](#), [6](#), [8](#), [10–22](#), [24–32](#), [34](#), [36](#), [38](#),
[40–42](#), [44–47](#), [49–52](#), [54](#), [56–58](#),
[60–65](#), [67](#), [69–71](#), [74–77](#), [79–85](#),
[87–98](#), [100–102](#), [104](#), [105](#), [107](#), [108](#),
[110–121](#), [123–130](#), [132](#), [133](#),
[135–138](#), [140](#), [142–157](#), [159](#), [160](#),
[162](#), [164–171](#), [174–177](#), [179–183](#),
[186–192](#), [195–198](#), [200](#), [201](#), [203](#),
[204](#), [206–222](#), [224](#), [225](#), [227](#),
[229–231](#), [233–239](#), [241–263](#), [265](#),
[266](#), [268–271](#), [273](#), [274](#), [276](#), [277](#),
[279–282](#), [284–286](#), [288](#), [289](#),
[291–297](#), [299](#), [300](#), [302](#), [303](#),
[305–308](#), [310](#), [311](#), [313](#), [315–320](#),
[322](#), [324](#), [325](#), [327–339](#), [341–346](#),
[348](#), [349](#), [351](#), [352](#), [354](#), [356](#), [357](#),
[359–363](#), [365–369](#), [371](#), [373](#), [374](#),
[376](#), [379–382](#), [384](#), [385](#)
- logical, [24–27](#), [29](#), [31](#), [32](#), [38](#), [40](#), [54](#), [56](#), [77](#),
[80](#), [83](#), [85](#), [88](#), [90](#), [93](#), [96](#), [98](#), [108](#),
[111](#), [113](#), [130](#), [133](#), [136](#), [138](#), [140](#),
[143](#), [177](#), [180](#), [182](#), [187](#), [189](#), [204](#),
[207](#), [209](#), [212–214](#), [219](#), [222](#), [225](#),
[227](#), [229](#), [255](#), [257](#), [263](#), [266](#), [269](#),
[280](#), [284](#), [300](#), [303](#), [306](#), [311](#), [320](#),
[322](#), [325](#), [342](#), [344](#), [352](#), [357](#), [374](#)
- logloss, [7](#), [9](#), [11](#), [14](#), [17](#), [20](#), [23](#), [27](#), [29](#), [30](#),
[32](#), [35](#), [37](#), [39](#), [41](#), [43](#), [45](#), [48](#), [50](#), [53](#),
[55](#), [57](#), [59](#), [61](#), [62](#), [64](#), [66](#), [68](#), [70](#), [73](#),
[75](#), [78](#), [80](#), [81](#), [83](#), [86](#), [88](#), [91](#), [94](#), [96](#),
[99](#), [101](#), [103](#), [104](#), [106](#), [109](#), [111](#),
[112](#), [114](#), [116](#), [118](#), [120](#), [122](#), [124](#),
[125](#), [127](#), [128](#), [131](#), [134](#), [136](#), [137](#),
[137](#), [145](#), [147](#), [149](#), [151](#), [152](#), [154](#),
[156](#), [158](#), [159](#), [161](#), [163](#), [165](#), [166](#),
[168](#), [170](#), [173](#), [175](#), [178](#), [180](#), [183](#),
[188](#), [189](#), [191](#), [192](#), [194](#), [196](#), [199](#),
[202](#), [205](#), [207](#), [208](#), [210](#), [216](#), [218](#),
[220](#), [223](#), [226](#), [228](#), [230](#), [232](#), [234](#),
[235](#), [237](#), [239](#), [240](#), [242](#), [245](#), [246](#),
[248](#), [250](#), [252](#), [253](#), [255](#), [257–259](#),
[261](#), [264](#), [267](#), [269](#), [270](#), [272](#), [275](#),
[278](#), [280](#), [283](#), [285](#), [287](#), [289](#), [290](#),
[292](#), [294](#), [296](#), [298](#), [301](#), [304](#), [306](#),
[307](#), [309](#), [312](#), [314](#), [316](#), [318](#), [321](#),
[328](#), [329](#), [331](#), [333](#), [336](#), [337](#), [340](#),
[342](#), [345](#), [347](#), [350](#), [353](#), [355](#), [357](#),
[358](#), [360](#), [361](#), [364](#), [366](#), [368](#), [370](#),
[372](#), [374](#), [375](#), [377](#), [381](#), [383](#), [385](#)
- logloss(), [137](#), [140](#), [142](#), [322](#), [324](#)
- logloss.factor, [138](#), [140](#)
- logloss.integer, [138](#), [142](#)
- maape, [7](#), [9](#), [11](#), [14](#), [17](#), [20](#), [23](#), [27](#), [30](#), [32](#), [35](#),
[37](#), [39](#), [41](#), [43](#), [45](#), [48](#), [50](#), [53](#), [55](#), [57](#),
[59](#), [61](#), [62](#), [64](#), [66](#), [68](#), [70](#), [73](#), [75](#), [78](#),
[81](#), [83](#), [86](#), [88](#), [91](#), [94](#), [96](#), [99](#), [101](#),
[104](#), [106](#), [109](#), [112](#), [114](#), [116](#), [118](#),
[120](#), [122](#), [125](#), [127](#), [128](#), [131](#), [134](#),
[137](#), [139](#), [141](#), [143](#), [144](#), [149–152](#),
[154](#), [156](#), [159](#), [161](#), [163](#), [165](#), [166](#),
[168](#), [170](#), [173](#), [175](#), [178](#), [180](#), [183](#),
[187–189](#), [192](#), [194](#), [196](#), [199](#), [202](#),
[205](#), [208](#), [210](#), [216–218](#), [220](#), [223](#),
[226](#), [228](#), [230](#), [232](#), [234](#), [235](#), [237](#),
[240](#), [242](#), [245](#), [246](#), [248](#), [250](#), [252](#),
[253](#), [255](#), [257](#), [259](#), [261](#), [264](#), [267](#),
[270](#), [272](#), [275](#), [278](#), [280](#), [283](#), [285](#),
[287](#), [290](#), [292–294](#), [296](#), [298](#), [301](#),
[304](#), [307](#), [309](#), [312](#), [314](#), [316](#), [318](#),
[321](#), [323](#), [325](#), [329](#), [331](#), [333](#), [336](#),
[337](#), [340](#), [342](#), [345](#), [347](#), [350](#), [353](#),
[355](#), [358](#), [360](#), [361](#), [364](#), [366](#), [368](#),
[370](#), [372](#), [375](#), [377](#), [381](#), [383](#), [385](#)
- maape(), [144](#), [146](#), [327](#)
- maape.numeric, [145](#), [146](#)
- mae, [7](#), [9](#), [11](#), [14](#), [17](#), [20](#), [23](#), [27](#), [30](#), [32](#), [35](#), [37](#),
[39](#), [41](#), [43](#), [45](#), [48](#), [50](#), [53](#), [55](#), [57](#), [59](#),
[61](#), [62](#), [64](#), [66](#), [68](#), [70](#), [73](#), [75](#), [78](#), [81](#),
[83](#), [86](#), [88](#), [91](#), [94](#), [96](#), [99](#), [101](#), [104](#),
[106](#), [109](#), [112](#), [114](#), [116](#), [118](#), [120](#),
[122](#), [125](#), [127](#), [128](#), [131](#), [134](#), [137](#),
[139](#), [141](#), [143](#), [145](#), [147](#), [148](#), [152](#),
[154](#), [156](#), [159](#), [161](#), [163](#), [165](#), [166](#),
[168](#), [170](#), [173](#), [175](#), [178](#), [180](#), [183](#),
[187–189](#), [192](#), [194](#), [196](#), [199](#), [202](#),
[205](#), [208](#), [210](#), [216–218](#), [220](#), [223](#),
[226](#), [228](#), [230](#), [232](#), [234](#), [235](#), [237](#),
[240](#), [242](#), [245](#), [246](#), [248](#), [250](#), [252](#),
[253](#), [255](#), [257](#), [259](#), [261](#), [264](#), [267](#),
[270](#), [272](#), [275](#), [278](#), [280](#), [283](#), [285](#),
[287](#), [290](#), [292–294](#), [296](#), [298](#), [301](#),
[304](#), [307](#), [309](#), [312](#), [314](#), [316](#), [318](#),

- 321, 323, 325, 328, 331, 333, 336,
337, 340, 342, 345, 347, 350, 353,
355, 358, 360, 361, 364, 366, 368,
370, 372, 375, 377, 381, 383, 385
- `mae()`, 148–150, 328
- `mae.numeric`, 148, 149
- `mape`, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35,
37, 39, 41, 43, 45, 48, 50, 53, 55, 57,
59, 61, 62, 64, 66, 68, 70, 73, 75, 78,
81, 83, 86, 88, 91, 94, 96, 99, 101,
104, 106, 109, 112, 114, 116, 118,
120, 122, 125, 127, 128, 131, 134,
137, 139, 141, 143, 145, 147,
149–151, 151, 156, 159, 161, 163,
165, 166, 168, 170, 173, 175, 178,
180, 183, 187–189, 192, 194, 196,
199, 202, 205, 208, 210, 216–218,
220, 223, 226, 228, 230, 232, 234,
235, 237, 240, 242, 245, 246, 248,
250, 252, 253, 255, 257, 259, 261,
264, 267, 270, 272, 275, 278, 280,
283, 285, 287, 290, 292–294, 296,
298, 301, 304, 307, 309, 312, 314,
316, 318, 321, 323, 325, 328, 329,
333, 336, 337, 340, 342, 345, 347,
350, 353, 355, 358, 360, 361, 364,
366, 368, 370, 372, 375, 377, 381,
383, 385
- `mape()`, 151, 153, 330
- `mape.numeric`, 152, 153
- `matrix`, 50, 52, 212–214, 289
- `mcc`, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35,
37, 39, 41, 43, 45, 48, 50, 53, 55, 57,
59, 61, 62, 64, 66, 68, 70, 73, 75, 78,
80, 81, 83, 86, 88, 91, 94, 96, 99,
101, 103, 104, 106, 109, 111, 112,
114, 116, 118, 120, 122, 124, 125,
127, 128, 131, 134, 136, 137, 139,
141, 143, 145, 147, 149, 151, 152,
154, 154, 163, 165, 166, 168, 170,
173, 175, 178, 180, 183, 188, 189,
191, 192, 194, 196, 199, 202, 205,
207, 208, 210, 216, 218, 220, 223,
226, 228, 230, 232, 234, 235, 237,
239, 240, 242, 245, 246, 248, 250,
252, 253, 255, 257, 259, 261, 264,
267, 269, 270, 272, 275, 278, 280,
283, 285, 287, 289, 290, 292, 294,
296, 298, 301, 304, 306, 307, 309,
312, 314, 316, 318, 321, 323, 325,
328, 329, 331, 336, 337, 340, 342,
345, 347, 350, 353, 355, 357, 358,
360, 361, 364, 366, 368, 370, 372,
374, 375, 377, 381, 383, 385
- `mcc()`, 154, 155, 157, 159, 160, 332
- `mcc.cmatrix`, 155, 157
- `mcc.cmatrix()`, 155, 158, 160, 333
- `mcc.factor`, 155, 159
- `mcc.factor()`, 155, 158, 160, 333
- `mean`, 244, 246, 366
- `mpe`, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37,
39, 41, 43, 45, 48, 50, 53, 55, 57, 59,
61, 62, 64, 66, 68, 70, 73, 75, 78, 81,
83, 86, 88, 91, 94, 96, 99, 101, 104,
106, 109, 112, 114, 116, 118, 120,
122, 125, 127, 128, 131, 134, 137,
139, 141, 143, 145, 147, 149–152,
154, 156, 159, 161, 162, 166, 168,
170, 173, 175, 178, 180, 183,
187–189, 192, 194, 196, 199, 202,
205, 208, 210, 216–218, 220, 223,
226, 228, 230, 232, 234, 235, 237,
240, 242, 245, 246, 248, 250, 252,
253, 255, 257, 259, 261, 264, 267,
270, 272, 275, 278, 280, 283, 285,
287, 290, 292–294, 296, 298, 301,
304, 307, 309, 312, 314, 316, 318,
321, 323, 325, 328, 329, 331, 333,
337, 340, 342, 345, 347, 350, 353,
355, 358, 360, 361, 364, 366, 368,
370, 372, 375, 377, 381, 383, 385
- `mpe()`, 162, 164, 334, 335
- `mpe.numeric`, 162, 163
- `mse`, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37,
39, 41, 43, 45, 48, 50, 53, 55, 57, 59,
61, 62, 64, 66, 68, 70, 73, 75, 78, 81,
83, 86, 88, 91, 94, 96, 99, 101, 104,
106, 109, 112, 114, 116, 118, 120,
122, 125, 127, 128, 131, 134, 137,
139, 141, 143, 145, 147, 149–152,
154, 156, 159, 161, 163, 165, 165,
170, 173, 175, 178, 180, 183,
187–189, 192, 194, 196, 199, 202,
205, 208, 210, 216–218, 220, 223,
226, 228, 230, 232, 234, 235, 237,
240, 242, 245, 246, 248, 250, 252,

- 253, 255, 257, 259, 261, 264, 267,
270, 272, 275, 278, 280, 283, 285,
287, 290, 292–294, 296, 298, 301,
304, 307, 309, 312, 314, 316, 318,
321, 323, 325, 328, 329, 331, 333,
336, 340, 342, 345, 347, 350, 353,
355, 358, 360, 361, 364, 366, 368,
370, 372, 375, 377, 381, 383, 385
- mse(), 66, 68, 165, 167, 295, 336
- mse.numeric, 166, 167
- NA, 5, 8, 10, 12, 15, 18, 21, 25, 27–32, 34, 36,
38, 40, 41, 44, 46, 48, 49, 51, 53, 54,
56, 58, 60, 61, 63, 65, 67, 69, 71, 74,
76, 77, 79–85, 87–90, 92–98, 100,
102, 104, 105, 107, 108, 110–113,
115, 117–119, 121, 123, 125–127,
129, 130, 132, 133, 135–137, 140,
142, 144, 146, 148–151, 153–155,
157, 159, 160, 162, 164, 165, 167,
169, 171, 174, 176, 177, 179–182,
186, 188, 190, 192, 195, 197, 200,
203, 204, 206–209, 211–219, 221,
222, 224, 225, 227, 229, 231, 233,
234, 236, 238, 241, 243–245, 247,
249–252, 254, 256, 258, 260, 262,
263, 265, 266, 268–270, 273, 276,
279–282, 284–286, 288, 291–294,
296, 299, 300, 302, 303, 305–308,
310, 311, 313, 315, 317, 319, 320,
322, 324, 327, 328, 330, 332,
334–336, 338, 341–345, 348, 351,
352, 354, 356–360, 362, 365, 367,
369, 371, 373–376, 379, 381, 382,
384
- nlr, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35,
37, 39, 41, 43, 45, 48, 50, 53, 55, 57,
59, 61, 62, 64, 66, 68, 70, 73, 75, 78,
80, 81, 83, 86, 88, 91, 94, 96, 99,
101, 103, 104, 106, 109, 111, 112,
114, 116, 118, 120, 122, 124, 125,
127, 128, 131, 134, 136, 137, 139,
141, 143, 145, 147, 149, 151, 152,
154, 156, 158, 159, 161, 163, 165,
166, 168, 169, 178, 180, 183, 188,
189, 191, 192, 194, 196, 199, 202,
205, 207, 208, 210, 216, 218, 220,
223, 226, 228, 230, 232, 234, 235,
237, 239, 240, 242, 245, 246, 248,
250, 252, 253, 255, 257, 259, 261,
264, 267, 269, 270, 272, 275, 278,
280, 283, 285, 287, 289, 290, 292,
294, 296, 298, 301, 304, 306, 307,
309, 312, 314, 316, 318, 321, 323,
325, 328, 329, 331, 333, 336, 337,
340, 345, 347, 350, 353, 355, 357,
358, 360, 361, 364, 366, 368, 370,
372, 374, 375, 377, 381, 383, 385
- nlp(), 169, 171, 174, 191, 194, 196, 298, 338,
347
- nlp.cmatrix, 170, 171
- nlp.cmatrix(), 169, 172, 174, 339
- nlp.factor, 170, 174
- nlp.factor(), 169, 172, 174, 339
- npv, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35,
37, 39, 41, 43, 45, 48, 50, 53, 55, 57,
59, 61, 62, 64, 66, 68, 70, 73, 75, 78,
80, 81, 83, 86, 88, 91, 94, 96, 99,
101, 103, 104, 106, 109, 111, 112,
114, 116, 118, 120, 122, 124, 125,
127, 128, 131, 134, 136, 137, 139,
141, 143, 145, 147, 149, 151, 152,
154, 156, 158, 159, 161, 163, 165,
166, 168, 170, 173, 175, 176, 188,
189, 191, 192, 194, 196, 199, 202,
205, 207, 208, 210, 216, 218, 220,
223, 226, 228, 230, 232, 234, 235,
237, 239, 240, 242, 245, 246, 248,
250, 252, 253, 255, 257–259, 261,
264, 267, 269, 270, 272, 275, 278,
280, 283, 285, 287, 289, 290, 292,
294, 296, 298, 301, 304, 306, 307,
309, 312, 314, 316, 318, 321, 323,
325, 328, 329, 331, 333, 336, 337,
340, 345, 347, 350, 353, 355, 357,
358, 360, 361, 364, 366, 368, 370,
372, 374, 375, 377, 381, 383, 385
- npv(), 176, 179, 181, 182, 341
- npv.cmatrix, 177, 179
- npv.cmatrix(), 177, 179, 182, 341
- npv.factor, 177, 181
- npv.factor(), 177, 179, 182, 341
- NULL, 186, 198, 201, 239, 242, 349, 363
- numeric, 198, 201, 239, 242, 349, 363
- obesity, 184
- OpenMP, 185
- openmp.off (OpenMP), 185

- openmp.on (OpenMP), 185
- openmp.threads (OpenMP), 185
- phi (mcc), 154
- phi(), 156, 158, 161, 333
- pinball, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 81, 83, 86, 88, 91, 94, 96, 99, 101, 104, 106, 109, 112, 114, 116, 118, 120, 122, 125, 127, 128, 131, 134, 137, 139, 141, 143, 145, 147, 149–152, 154, 156, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 181, 183, 186, 192, 194, 196, 199, 202, 205, 208, 210, 216–218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 240, 242, 245, 246, 248, 250, 252, 253, 255, 258, 259, 261, 264, 267, 270, 272, 275, 278, 280, 283, 285, 287, 290, 292–294, 296, 298, 301, 304, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 334, 336, 337, 340, 342, 347, 350, 353, 355, 358, 360, 361, 364, 366, 368, 370, 372, 375, 377, 381, 383, 385
- pinball(), 186, 188, 343, 344
- pinball.numeric, 187, 188
- plot, 13, 16, 19, 21, 273, 276
- plr, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 81, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 112, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 137, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 181, 183, 188, 189, 190, 199, 202, 205, 207, 208, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 240, 242, 245, 246, 248, 250, 252, 253, 255, 257–259, 261, 264, 267, 269, 270, 272, 275, 278, 280, 283, 285, 287, 289, 290, 292, 294, 296, 298, 299, 301, 304, 306, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 334, 336, 337, 340, 342, 345, 347, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- plr(), 170, 173, 175, 190, 192, 195, 296, 339, 345
- plr.cmatrix, 191, 192
- plr.cmatrix(), 191, 193, 195, 297, 346
- plr.factor, 191, 195
- plr.factor(), 191, 193, 195, 297, 346
- ppv (precision), 203
- ppv(), 204, 207, 210, 352
- pr.curve, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 81, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 112, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 137, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 181, 183, 188, 189, 191, 192, 194, 196, 197, 205, 207, 208, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 240, 242, 245, 246, 248, 250, 252, 253, 255, 257–259, 261, 264, 267, 269, 270, 272, 275, 278, 280, 283, 285, 287, 289, 290, 292, 294, 296, 298, 301, 304, 306, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 334, 336, 337, 340, 342, 345, 347, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- pr.curve(), 13, 16, 197, 200, 273, 348
- pr.curve.factor, 198, 200
- precision, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 81, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 112, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 137, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178,

- 180, 181, 183, 188, 189, 191, 192, 194, 196, 199, 202, 203, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 239, 240, 242, 245, 246, 248, 250, 252, 253, 255, 257–259, 261, 264, 267, 269, 270, 272, 275, 278, 280, 283, 285, 287, 289, 290, 292, 294, 296, 298, 301, 304, 306, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 334, 336, 337, 340, 342, 345, 347, 350, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385*
- precision(), 198, 201, 203, 206, 208, 209, 349, 351*
- precision.cmatrix, 204, 205*
- precision.cmatrix(), 203, 206, 209, 352*
- precision.factor, 204, 208*
- precision.factor(), 203, 206, 209, 352*
- preorder, 211, 214*
- preorder(), 211, 212*
- preorder.matrix, 212, 212*
- presort, 212, 213, 213*
- presort(), 213, 214*
- presort.matrix, 213, 214*
- rae, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 81, 83, 86, 88, 91, 94, 96, 99, 101, 104, 106, 109, 112, 114, 116, 118, 120, 122, 125, 127, 128, 131, 134, 137, 139, 141, 143, 145, 147, 149–152, 154, 156, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 181, 183, 187–189, 192, 194, 196, 199, 202, 205, 208, 210, 215, 220, 223, 226, 228, 230, 232, 234, 235, 237, 240, 242, 245, 246, 248, 250, 252, 253, 255, 258, 259, 261, 264, 267, 270, 272, 275, 278, 280, 283, 285, 287, 290, 292–294, 296, 298, 301, 304, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 334, 336, 337, 340, 342, 345, 347, 350, 353, 358, 360, 361, 364, 366, 368, 370, 372, 375, 377, 381, 383, 385*
- rae(), 215–217, 354*
- rae.numeric, 215, 216*
- range, 244, 246, 366*
- recall, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 81, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 112, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 137, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 181, 183, 188, 189, 191, 192, 194, 196, 199, 202, 205, 207, 208, 210, 216, 218, 218, 228, 230, 232, 234, 235, 237, 239, 240, 242, 245, 246, 248, 250, 252, 253, 255, 257–259, 261, 264, 267, 269, 270, 272, 275, 278, 280, 283, 285, 287, 289, 290, 292, 294, 296, 298, 301, 304, 306, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 334, 336, 337, 340, 342, 345, 347, 350, 353, 355, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385*
- recall(), 198, 201, 218, 221, 224, 349, 356*
- recall.cmatrix, 219, 221*
- recall.cmatrix(), 219, 222, 224, 356*
- recall.factor, 219, 224*
- recall.factor(), 219, 222, 224, 356*
- relative.entropy, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 81, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 112, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 137, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 181, 183, 188, 189, 191, 192, 194, 196, 199, 202, 205, 207, 208, 210, 216, 218, 220, 223, 226, 226, 232, 234, 235, 237, 239, 240, 242, 245, 246, 248, 250, 252, 253, 255, 257–259, 261, 264, 267, 269, 270, 272, 275, 278, 280, 283,*

- 285, 287, 289, 290, 292, 294, 296, 298, 301, 304, 306, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 333, 334, 336, 337, 340, 342, 345, 347, 350, 353, 355, 357, 358, 360, 361, 364, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- `relative.entropy()`, 226, 227, 229
- `relative.entropy.matrix`, 227, 229
- `rmse`, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 81, 83, 86, 88, 91, 94, 96, 99, 101, 104, 106, 109, 112, 114, 116, 118, 120, 122, 125, 127, 128, 131, 134, 137, 139, 141, 143, 145, 147, 149–152, 154, 156, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 181, 183, 187–189, 192, 194, 196, 199, 202, 205, 208, 210, 216–218, 220, 223, 226, 228, 230, 231, 235, 237, 240, 242, 245, 246, 248, 250, 252, 253, 255, 258, 259, 261, 264, 267, 270, 272, 275, 278, 280, 283, 285, 287, 290, 292–294, 296, 298, 301, 304, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 329, 331, 334, 336, 337, 340, 342, 345, 347, 350, 353, 355, 358, 361, 362, 364, 366, 368, 370, 372, 375, 377, 381, 383, 385
- `rmse()`, 231, 233, 358, 359
- `rmse.numeric`, 231, 232
- `rmsle`, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 81, 83, 86, 88, 91, 94, 96, 99, 101, 104, 106, 109, 112, 114, 116, 118, 120, 122, 125, 127, 128, 131, 134, 137, 139, 141, 143, 145, 147, 149–152, 154, 156, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 181, 183, 187–189, 192, 194, 196, 199, 202, 205, 208, 210, 216–218, 220, 223, 226, 228, 230, 232, 234, 237, 240, 242, 245, 246, 248, 250, 252, 253, 255, 258, 259, 261, 264, 267, 270, 272, 275, 278, 280, 283, 285, 287, 290, 292–294, 296, 298, 301, 304, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328–331, 334, 336–338, 340, 342, 345, 347, 350, 353, 355, 358, 360, 364, 366, 368, 370, 372, 375, 377, 381, 383, 385
- `rmsle()`, 234, 236, 360
- `rmsle.numeric`, 235, 236
- `roc.curve`, 7, 9, 11, 14, 17, 20, 23, 27, 29, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 80, 81, 83, 86, 88, 91, 94, 96, 99, 101, 103, 104, 106, 109, 111, 112, 114, 116, 118, 120, 122, 124, 125, 127, 128, 131, 134, 136, 137, 139, 141, 143, 145, 147, 149, 151, 152, 154, 156, 158, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 180, 181, 183, 188, 189, 191, 192, 194, 196, 199, 202, 205, 207, 208, 210, 216, 218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 238, 245, 246, 248, 250, 252, 253, 255, 257–259, 261, 264, 267, 269, 270, 272, 275, 278, 280, 283, 285, 287, 289, 290, 292, 294, 296, 298, 301, 304, 306, 307, 309, 312, 314, 316, 318, 321, 323, 325, 328, 330, 331, 333, 334, 336, 338, 340, 342, 345, 347, 350, 353, 355, 357, 358, 360, 362, 366, 368, 370, 372, 374, 375, 377, 381, 383, 385
- `roc.curve()`, 19, 21, 238, 241, 276, 362
- `roc.curve.factor`, 239, 240
- `rrmse`, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37, 39, 41, 43, 45, 48, 50, 53, 55, 57, 59, 61, 62, 64, 66, 68, 70, 73, 75, 78, 81, 83, 86, 88, 91, 94, 96, 99, 101, 104, 106, 109, 112, 114, 116, 118, 120, 122, 125, 127, 128, 131, 134, 137, 139, 141, 143, 145, 147, 149–152, 154, 156, 159, 161, 163, 165, 166, 168, 170, 173, 175, 178, 181, 183, 187–189, 192, 194, 196, 199, 202, 205, 208, 210, 216–218, 220, 223, 226, 228, 230, 232, 234, 235, 237, 240, 242, 243, 248, 250, 252, 253, 255, 258, 259, 261, 264,

- 267, 270, 272, 275, 278, 280, 283,
285, 287, 290, 292–294, 296, 298,
301, 304, 307, 309, 312, 314, 316,
318, 321, 323, 325, 328–331, 334,
336–338, 340, 342, 345, 347, 350,
353, 355, 358, 360–362, 364, 368,
370, 372, 375, 377, 381, 383, 385
- `rrmse()`, 243–245, 365
- `rrmse.numeric`, 244, 245
- `rrse`, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35,
37, 39, 41, 43, 45, 48, 50, 53, 55, 57,
59, 61, 62, 64, 66, 68, 70, 73, 75, 78,
81, 83, 86, 88, 91, 94, 96, 99, 101,
104, 106, 109, 112, 114, 116, 118,
120, 122, 125, 127, 128, 131, 134,
137, 139, 141, 143, 145, 147,
149–152, 154, 156, 159, 161, 163,
165, 166, 168, 170, 173, 175, 178,
181, 183, 187–189, 192, 194, 196,
199, 202, 205, 208, 210, 216–218,
220, 223, 226, 228, 230, 232, 234,
235, 237, 240, 242, 245, 246, 247,
252, 253, 255, 258, 259, 261, 264,
267, 270, 272, 275, 278, 280, 283,
285, 287, 290, 292–294, 296, 298,
301, 304, 307, 309, 312, 314, 316,
318, 321, 323, 325, 328–331, 334,
336–338, 340, 342, 345, 347, 350,
353, 355, 358, 360–362, 364, 366,
370, 372, 375, 377, 381, 383, 385
- `rrse()`, 247, 249, 367
- `rrse.numeric`, 248, 249
- `rsq`, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35, 37,
39, 41, 43, 45, 48, 50, 53, 55, 57, 59,
61, 62, 64, 66, 68, 70, 73, 75, 78, 81,
83, 86, 88, 91, 94, 96, 99, 101, 104,
106, 109, 112, 114, 116, 118, 120,
122, 125, 127, 128, 131, 134, 137,
139, 141, 143, 145, 147, 149–152,
154, 156, 159, 161, 163, 165, 166,
168, 170, 173, 175, 178, 181, 183,
187–189, 192, 194, 196, 199, 202,
205, 208, 210, 216–218, 220, 223,
226, 228, 230, 232, 234, 235, 237,
240, 242, 245, 246, 248, 250, 250,
255, 258, 259, 261, 264, 267, 270,
272, 275, 278, 280, 283, 285, 287,
290, 292–294, 296, 298, 301, 304,
307, 309, 312, 314, 316, 318, 321,
323, 325, 328–331, 334, 336–338,
340, 342, 345, 347, 350, 353, 355,
358, 360–362, 364, 366, 368, 372,
375, 377, 381, 383, 385
- `rsq()`, 250–252, 369
- `rsq.numeric`, 251, 252
- `selectivity (specificity)`, 262
- `selectivity()`, 263, 266, 269, 374
- `sensitivity (recall)`, 218
- `sensitivity()`, 220, 222, 225, 357
- `shannon.entropy`, 7, 9, 11, 14, 17, 20, 23, 27,
29, 30, 32, 35, 37, 39, 41, 43, 45, 48,
50, 53, 55, 57, 59, 61, 62, 64, 66, 68,
70, 73, 75, 78, 80, 81, 83, 86, 88, 91,
94, 96, 99, 101, 103, 104, 106, 109,
111, 112, 114, 116, 118, 120, 122,
124, 125, 127, 128, 131, 134, 136,
137, 139, 141, 143, 145, 147, 149,
151, 152, 154, 156, 158, 159, 161,
163, 165, 166, 168, 170, 173, 175,
178, 180, 181, 183, 188, 189, 191,
192, 194, 196, 199, 202, 205, 207,
208, 210, 216, 218, 220, 223, 226,
228, 230, 232, 234, 235, 237, 239,
240, 242, 245, 246, 248, 250, 252,
253, 254, 259, 261, 264, 267, 269,
270, 272, 275, 278, 280, 283, 285,
287, 289, 290, 292, 294, 296, 298,
301, 304, 306, 307, 309, 312, 314,
316, 318, 321, 323, 325, 328, 330,
331, 333, 334, 336, 338, 340, 342,
345, 347, 350, 353, 355, 357, 358,
360, 362, 364, 366, 368, 370, 372,
374, 375, 377, 381, 383, 385
- `shannon.entropy()`, 254, 256
- `shannon.entropy.matrix`, 255, 256
- `smape`, 7, 9, 11, 14, 17, 20, 23, 27, 30, 32, 35,
37, 39, 41, 43, 45, 48, 50, 53, 55, 57,
59, 61, 62, 64, 66, 68, 70, 73, 75, 78,
81, 83, 86, 88, 91, 94, 96, 99, 101,
104, 106, 109, 112, 114, 116, 118,
120, 122, 125, 127, 128, 131, 134,
137, 139, 141, 143, 145, 147,
149–152, 154, 156, 159, 161, 163,
165, 166, 168, 170, 173, 175, 178,
181, 183, 187–189, 192, 194, 196,
199, 202, 205, 208, 210, 216–218,

- 220, 223, 226, 228, 230, 232, 234,
235, 237, 240, 242, 245, 246, 248,
250, 252, 253, 255, 258, 258, 264,
267, 270, 272, 275, 278, 280, 283,
285, 287, 290, 292–294, 296, 298,
301, 304, 307, 309, 312, 314, 316,
318, 321, 323, 325, 328–331, 334,
336–338, 340, 342, 345, 347, 350,
353, 355, 358, 360–362, 364, 366,
368, 370, 375, 377, 381, 383, 385
- `smape()`, 258, 260, 371
- `smape.numeric`, 259, 260
- `specificity`, 7, 9, 11, 14, 17, 20, 23, 27, 29,
30, 32, 35, 37, 39, 41, 43, 45, 48, 50,
53, 55, 57, 59, 61, 62, 64, 66, 68, 70,
73, 75, 78, 80, 81, 83, 86, 88, 91, 94,
96, 99, 101, 103, 104, 106, 109, 111,
112, 114, 116, 118, 120, 122, 124,
125, 127, 128, 131, 134, 136, 137,
139, 141, 143, 145, 147, 149, 151,
152, 154, 156, 158, 159, 161, 163,
165, 166, 168, 170, 173, 175, 178,
180, 181, 183, 188, 189, 191, 192,
194, 196, 199, 202, 205, 207, 208,
210, 216, 218, 220, 223, 226, 228,
230, 232, 234, 235, 237, 239, 240,
242, 245, 246, 248, 250, 252, 253,
255, 257–259, 261, 262, 272, 275,
278, 280, 283, 285, 287, 289, 290,
292, 294, 296, 298, 301, 304, 306,
307, 309, 312, 314, 316, 318, 321,
323, 325, 328, 330, 331, 333, 334,
336, 338, 340, 342, 345, 347, 350,
353, 355, 357, 358, 360, 362, 364,
366, 368, 370, 372, 377, 381, 383,
385
- `specificity()`, 262, 265, 268, 373
- `specificity.cmatrix`, 263, 265
- `specificity.cmatrix()`, 262, 265, 268, 373
- `specificity.factor`, 263, 267
- `specificity.factor()`, 262, 265, 268, 373
- `tnr (specificity)`, 262
- `tnr()`, 263, 266, 269, 374
- `tpr (recall)`, 218
- `tpr()`, 220, 222, 225, 239, 242, 357, 363
- `TRUE`, 24–27, 29, 31, 32, 38, 40, 54, 56, 77, 80,
83, 85, 88, 90, 93, 96, 98, 108, 111,
113, 130, 133, 136, 138, 140, 143,
177, 180, 182, 187, 189, 204, 207,
209, 219, 222, 225, 227, 229, 255,
257, 263, 266, 269, 280, 284, 300,
303, 306, 311, 320, 322, 325, 342,
344, 352, 357, 374
- `try()`, 5, 8, 10, 12, 15, 18, 21, 25, 28, 31, 34,
36, 38, 40, 41, 44, 46, 49, 51, 54, 56,
58, 60, 61, 63, 65, 67, 69, 71, 74, 76,
79, 81, 84, 87, 89, 92, 95, 97, 100,
102, 105, 107, 110, 112, 115, 117,
119, 121, 123, 125, 127, 129, 132,
135, 137, 140, 142, 144, 146, 148,
150, 151, 153, 155, 157, 160, 162,
164, 165, 167, 169, 171, 174, 176,
179, 181, 186, 188, 190, 192, 195,
197, 200, 203, 206, 208, 211–215,
217, 218, 221, 224, 227, 229, 231,
233, 234, 236, 238, 241, 244, 245,
247, 249, 251, 252, 254, 256, 258,
260, 262, 265, 268, 270, 273, 276,
279, 281, 284, 286, 288, 291, 292,
294, 296, 299, 302, 305, 308, 310,
313, 315, 317, 319, 322, 324, 327,
328, 330, 332, 335, 336, 338, 341,
343, 345, 348, 351, 354, 356, 359,
360, 362, 365, 367, 369, 371, 373,
376, 379, 382, 384
- `tryCatch()`, 5, 8, 10, 12, 15, 18, 21, 25, 28,
31, 34, 36, 38, 40, 41, 44, 46, 49, 51,
54, 56, 58, 60, 61, 63, 65, 67, 69, 71,
74, 76, 79, 81, 84, 87, 89, 92, 95, 97,
100, 102, 105, 107, 110, 112, 115,
117, 119, 121, 123, 125, 127, 129,
132, 135, 137, 140, 142, 144, 146,
148, 150, 151, 153, 155, 157, 160,
162, 164, 165, 167, 169, 171, 174,
176, 179, 181, 186, 188, 190, 192,
195, 197, 200, 203, 206, 208,
211–215, 217, 218, 221, 224, 227,
229, 231, 233, 234, 236, 238, 241,
244, 245, 247, 249, 251, 252, 254,
256, 258, 260, 262, 265, 268, 270,
273, 276, 279, 281, 284, 286, 288,
291, 292, 294, 296, 299, 302, 305,
308, 310, 313, 315, 317, 319, 322,
324, 327, 328, 330, 332, 335, 336,
338, 341, 343, 345, 348, 351, 354,
356, 359, 360, 362, 365, 367, 369,

- [371, 373, 376, 379, 382, 384](#)
- [tscore\(jaccard\), 129](#)
- [tscore\(\), 130, 133, 136, 320](#)
- [weighted.accuracy\(accuracy\), 5](#)
- [weighted.accuracy.factor, 6, 270](#)
- [weighted.auc.pr.curve\(auc.pr.curve\), 12](#)
- [weighted.auc.pr.curve.factor, 13, 273](#)
- [weighted.auc.roc.curve\(auc.roc.curve\), 18](#)
- [weighted.auc.roc.curve.factor, 19, 276](#)
- [weighted.baccracy\(baccracy\), 25](#)
- [weighted.baccracy.factor, 26, 279](#)
- [weighted.brier.score\(brier.score\), 34](#)
- [weighted.brier.score.matrix, 34, 281](#)
- [weighted.ccc\(ccc\), 38](#)
- [weighted.ccc.numeric, 38, 283](#)
- [weighted.ckappa\(ckappa\), 41](#)
- [weighted.ckappa.factor, 42, 285](#)
- [weighted.cmatrix\(cmatrix\), 48](#)
- [weighted.cmatrix.factor, 49, 288](#)
- [weighted.csi\(jaccard\), 129](#)
- [weighted.deviance.gamma](#)
[\(deviance.gamma\), 58](#)
- [weighted.deviance.gamma.numeric, 58, 290](#)
- [weighted.deviance.poisson](#)
[\(deviance.poisson\), 61](#)
- [weighted.deviance.poisson.numeric, 62, 292](#)
- [weighted.deviance.tweedie](#)
[\(deviance.tweedie\), 65](#)
- [weighted.deviance.tweedie.numeric, 65, 294](#)
- [weighted.dor\(plr\), 190](#)
- [weighted.dor.factor, 70, 296](#)
- [weighted.fallout\(fpr\), 107](#)
- [weighted.fbeta\(fbeta\), 76](#)
- [weighted.fbeta.factor, 77, 299](#)
- [weighted.fdr\(fdr\), 84](#)
- [weighted.fdr.factor, 85, 302](#)
- [weighted.fer\(fer\), 92](#)
- [weighted.fer.factor, 93, 305](#)
- [weighted.fmi\(fmi\), 100](#)
- [weighted.fmi.factor, 101, 307](#)
- [weighted.fpr\(fpr\), 107](#)
- [weighted.fpr.factor, 108, 310](#)
- [weighted.gmse\(gmse\), 115](#)
- [weighted.gmse.numeric, 116, 313](#)
- [weighted.hammingloss\(hammingloss\), 118](#)
- [weighted.hammingloss.factor, 119, 314](#)
- [weighted.huberloss\(huberloss\), 125](#)
- [weighted.huberloss.numeric, 126, 317](#)
- [weighted.jaccard\(jaccard\), 129](#)
- [weighted.jaccard.factor, 130, 319](#)
- [weighted.logloss\(logloss\), 137](#)
- [weighted.logloss.factor, 138, 322](#)
- [weighted.logloss.integer, 138, 324](#)
- [weighted.maape\(maape\), 144](#)
- [weighted.maape.numeric, 145, 326](#)
- [weighted.mae\(mae\), 148](#)
- [weighted.mae.numeric, 148, 328](#)
- [weighted.mape\(mape\), 151](#)
- [weighted.mape.numeric, 152, 330](#)
- [weighted.mcc\(mcc\), 154](#)
- [weighted.mcc.factor, 155, 332](#)
- [weighted.mpe\(mpe\), 162](#)
- [weighted.mpe.numeric, 162, 334](#)
- [weighted.mse\(mse\), 165](#)
- [weighted.mse.numeric, 166, 336](#)
- [weighted.nlr\(nlr\), 169](#)
- [weighted.nlr.factor, 170, 338](#)
- [weighted.npv\(npv\), 176](#)
- [weighted.npv.factor, 177, 340](#)
- [weighted.phi\(mcc\), 154](#)
- [weighted.pinball\(pinball\), 186](#)
- [weighted.pinball.numeric, 187, 343](#)
- [weighted.plr\(plr\), 190](#)
- [weighted.plr.factor, 191, 345](#)
- [weighted.ppv\(precision\), 203](#)
- [weighted.pr.curve\(pr.curve\), 197](#)
- [weighted.pr.curve.factor, 198, 348](#)
- [weighted.precision\(precision\), 203](#)
- [weighted.precision.factor, 204, 351](#)
- [weighted.rae\(rae\), 215](#)
- [weighted.rae.numeric, 215, 354](#)
- [weighted.recall\(recall\), 218](#)
- [weighted.recall.factor, 219, 355](#)
- [weighted.rmse\(rmse\), 231](#)
- [weighted.rmse.numeric, 231, 358](#)
- [weighted.rmsle\(rmsle\), 234](#)
- [weighted.rmsle.numeric, 235, 360](#)
- [weighted.roc.curve\(roc.curve\), 238](#)
- [weighted.roc.curve.factor, 239, 362](#)
- [weighted.rrmse\(rrmse\), 243](#)
- [weighted.rrmse.numeric, 244, 365](#)
- [weighted.rrse\(rrse\), 247](#)

`weighted.rrse.numeric`, 248, 367
`weighted.rsq(rsq)`, 250
`weighted.rsq.numeric`, 251, 369
`weighted.selectivity(specificity)`, 262
`weighted.sensitivity(recall)`, 218
`weighted.smape(smape)`, 258
`weighted.smape.numeric`, 259, 371
`weighted.specificity(specificity)`, 262
`weighted.specificity.factor`, 263, 372
`weighted.tnr(specificity)`, 262
`weighted.tpr(recall)`, 218
`weighted.tscore(jaccard)`, 129
`weighted.zerooneloss(zerooneloss)`, 379
`weighted.zerooneloss.factor`, 375, 380
`wine.quality`, 378

`zerooneloss`, 7, 9, 11, 14, 17, 20, 23, 27, 29,
30, 32, 35, 37, 39, 41, 43, 45, 48, 50,
53, 55, 57, 59, 61, 62, 64, 66, 68, 70,
73, 75, 78, 80, 81, 83, 86, 88, 91, 94,
96, 99, 101, 103, 104, 106, 109, 111,
112, 114, 116, 118, 120, 122, 124,
125, 127, 128, 131, 134, 136, 137,
139, 141, 143, 145, 147, 149, 151,
152, 154, 156, 158, 159, 161, 163,
165, 166, 168, 170, 173, 175, 178,
180, 181, 183, 188, 189, 191, 192,
194, 196, 199, 202, 205, 207, 208,
210, 216, 218, 220, 223, 226, 228,
230, 232, 234, 235, 237, 239, 240,
242, 245, 246, 248, 250, 252, 253,
255, 257–259, 261, 264, 267, 269,
270, 272, 275, 278, 280, 283, 285,
287, 289, 290, 292, 294, 296, 298,
301, 304, 306, 307, 309, 312, 314,
316, 318, 321, 323, 325, 328, 330,
331, 333, 334, 336, 338, 340, 342,
345, 347, 350, 353, 355, 357, 358,
360, 362, 364, 366, 368, 370, 372,
374, 375, 379

`zerooneloss()`, 375, 376, 379, 381, 382, 384
`zerooneloss.cmatrix`, 380, 381
`zerooneloss.cmatrix()`, 376, 380, 382, 384
`zerooneloss.factor`, 380, 384
`zerooneloss.factor()`, 376, 380, 382, 384