

# Package ‘MetricsWeighted’

June 5, 2023

**Title** Weighted Metrics and Performance Measures for Machine Learning

**Version** 1.0.2

**Description** Provides weighted versions of several metrics and performance measures used in machine learning, including average unit deviances of the Bernoulli, Tweedie, Poisson, and Gamma distributions, see Jorgensen B. (1997, ISBN: 978-0412997112). The package also contains a weighted version of generalized R-squared, see e.g. Cohen, J. et al. (2002, ISBN: 978-0805822236). Furthermore, 'dplyr' chains are supported.

**License** GPL (>= 2)

**Depends** R (>= 3.1.0)

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Imports** graphics, stats

**URL** <https://github.com/mayer79/MetricsWeighted>

**BugReports** <https://github.com/mayer79/MetricsWeighted/issues>

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

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**NeedsCompilation** no

**Author** Michael Mayer [aut, cre],  
Christian Lorentzen [ctb]

**Maintainer** Michael Mayer <mayermichael79@gmail.com>

**Repository** CRAN

**Date/Publication** 2023-06-05 15:50:07 UTC

**R topics documented:**

classification . . . . .	2
elementary_score . . . . .	4
multi_metric . . . . .	5
murphy_diagram . . . . .	6
performance . . . . .	7
regression . . . . .	9
rsquared . . . . .	11
weighted_cor . . . . .	13
weighted_mean . . . . .	14
weighted_median . . . . .	14
weighted_quantile . . . . .	15
weighted_var . . . . .	16
<b>Index</b>	<b>18</b>

---

classification	<i>Classification Metrics</i>
----------------	-------------------------------

---

**Description**

Weighted versions of non-probabilistic and probabilistic classification metrics:

- `accuracy()`: Accuracy (higher is better).
- `classification_error()`: Classification error = 1 - Accuracy (lower is better).
- `precision()`: Precision (higher is better).
- `recall()`: Recall (higher is better).
- `f1_score()`: F1 Score. Harmonic mean of precision and recall (higher is better).
- `AUC()`: Area under the ROC (higher is better).
- `gini_coefficient()`: Gini coefficient, equivalent to  $2 \cdot \text{AUC} - 1$ . Up to ties in predicted, equivalent to Somer's D (higher is better).
- `deviance_bernoulli()`: Average Bernoulli deviance. Equals twice the log loss/binary cross entropy (smaller is better).
- `logLoss()`: Log loss/binary cross entropy. Equals half the average Bernoulli deviance (smaller is better).

**Usage**

```
accuracy(actual, predicted, w = NULL, ...)
```

```
classification_error(actual, predicted, w = NULL, ...)
```

```
precision(actual, predicted, w = NULL, ...)
```

```
recall(actual, predicted, w = NULL, ...)
```

```
f1_score(actual, predicted, w = NULL, ...)  
AUC(actual, predicted, w = NULL, ...)  
gini_coefficient(actual, predicted, w = NULL, ...)  
deviance_bernoulli(actual, predicted, w = NULL, ...)  
logLoss(actual, predicted, w = NULL, ...)
```

### Arguments

actual	Observed values.
predicted	Predicted values.
w	Optional case weights.
...	Further arguments passed to <code>weighted_mean()</code> (no effect for <code>AUC()</code> and <code>gini_coefficient()</code> ).

### Details

Note that the function `AUC()` was originally modified from the `glmnet` package to ensure deterministic results. The unweighted version can be different from the weighted one with unit weights due to ties in predicted.

### Value

A numeric vector of length one.

### Input ranges

- For `precision()`, `recall()`, and `f1_score()`: The actual and predicted values need to be in  $\{0, 1\}$ .
- For `accuracy()` and `classification_error()`: Any discrete input.
- For `AUC()` and `gini_coefficient()`: Only actual must be in  $\{0, 1\}$ .
- For `deviance_bernoulli()` and `logLoss()`: The values of actual must be in  $\{0, 1\}$ , while predicted must be in the open interval  $(0, 1)$ .

### Examples

```
y <- c(0, 0, 1, 1)  
pred <- c(0, 0, 1, 0)  
w <- y * 2  
  
accuracy(y, pred)  
classification_error(y, pred, w = w)  
precision(y, pred, w = w)  
recall(y, pred, w = w)  
f1_score(y, pred, w = w)
```

```

y2 <- c(0, 1, 0, 1)
pred2 <- c(0.1, 0.1, 0.9, 0.8)
w2 <- 1:4

AUC(y2, pred2)
AUC(y2, pred2, w = rep(1, 4)) # Different due to ties in predicted

gini_coefficient(y2, pred2, w = w2)
logLoss(y2, pred2, w = w2)
deviance_bernoulli(y2, pred2, w = w2)

```

---

 elementary\_score

*Elementary Scoring Function for Expectiles and Quantiles*


---

### Description

Weighted average of the elementary scoring function for expectiles or quantiles at level  $\alpha$  with parameter  $\theta$ , see reference below. Every choice of  $\theta$  gives a scoring function consistent for the expectile or quantile at level  $\alpha$ . Note that the expectile at level  $\alpha = 0.5$  is the expectation (mean). The smaller the score, the better.

### Usage

```

elementary_score_expectile(
  actual,
  predicted,
  w = NULL,
  alpha = 0.5,
  theta = 0,
  ...
)

```

```

elementary_score_quantile(
  actual,
  predicted,
  w = NULL,
  alpha = 0.5,
  theta = 0,
  ...
)

```

### Arguments

actual	Observed values.
predicted	Predicted values.
w	Optional case weights.

alpha	Level of expectile or quantile. The default alpha = 0.5 corresponds to the expectation/median.
theta	Evaluation point.
...	Further arguments passed to <code>weighted_mean()</code> .

**Value**

A numeric vector of length one.

**References**

Ehm, W., Gneiting, T., Jordan, A. and Krüger, F. (2016), Of quantiles and expectiles: consistent scoring functions, Choquet representations and forecast rankings. *J. R. Stat. Soc. B*, 78: 505-562, <[doi.org/10.1111/rssb.12154](https://doi.org/10.1111/rssb.12154)>.

**See Also**

[murphy\\_diagram\(\)](#)

**Examples**

```
elementary_score_expectile(1:10, c(1:9, 12), alpha = 0.5, theta = 11)
elementary_score_quantile(1:10, c(1:9, 12), alpha = 0.5, theta = 11)
```

---

multi_metric	<i>Multiple Metrics</i>
--------------	-------------------------

---

**Description**

Provides a way to create a list of metrics/performance measures from a parametrized function like the Tweedie deviance or the elementary scoring functions for expectiles.

**Usage**

```
multi_metric(fun, ...)
```

**Arguments**

fun	A metric/performance measure with additional parameter to be varied.
...	Further arguments passed to <code>fun()</code> , including one varying parameter (specified by a vector).

**Value**

A named list of functions.

**See Also**[performance\(\)](#)**Examples**

```

data <- data.frame(act = 1:10, pred = c(1:9, 12))
multi <- multi_metric(fun = deviance_tweedie, tweedie_p = c(0, seq(1, 3, by = 0.1)))
performance(data, actual = "act", predicted = "pred", metrics = multi)
multi <- multi_metric(
  fun = r_squared,
  deviance_function = deviance_tweedie, tweedie_p = c(0, seq(1, 3, by = 0.1))
)
performance(data, actual = "act", predicted = "pred", metrics = multi)
multi <- multi_metric(fun = elementary_score_expectile, theta = 1:11, alpha = 0.1)
performance(data, actual = "act", predicted = "pred", metrics = multi, key = "theta")
multi <- multi_metric(fun = elementary_score_expectile, theta = 1:11, alpha = 0.5)
performance(data, actual = "act", predicted = "pred", metrics = multi, key = "theta")

```

murphy\_diagram

*Murphy diagram***Description**

Murphy diagram of the elementary scoring function for expectiles/quantiles at level  $\alpha$  for different values of  $\theta$ . Can be used to study and compare performance of one or multiple models.

**Usage**

```

murphy_diagram(
  actual,
  predicted,
  w = NULL,
  alpha = 0.5,
  theta = seq(-2, 2, length.out = 100L),
  functional = c("expectile", "quantile"),
  plot = TRUE,
  ...
)

```

**Arguments**

actual	Observed values.
predicted	Predicted values.
w	Optional case weights.
alpha	Level of expectile or quantile. The default $\alpha = 0.5$ corresponds to the expectation/median.
theta	Vector of evaluation points.

functional	Either "expectile" or "quantile".
plot	Should a plot be returned (default is TRUE)? If FALSE, a data.frame containing the results.
...	Further arguments passed to <code>graphics::matplot()</code> .

### Details

If the plot needs to be customized, set `plot = FALSE` to get the resulting data instead of the plot.

### Value

The result of `graphics::matplot()` or a data.frame containing the results.

### References

Ehm, W., Gneiting, T., Jordan, A. and Krüger, F. (2016), Of quantiles and expectiles: consistent scoring functions, Choquet representations and forecast rankings. *J. R. Stat. Soc. B*, 78: 505-562, <[doi.org/10.1111/rssb.12154](https://doi.org/10.1111/rssb.12154)>.

### See Also

`elementary_score()`

### Examples

```
y <- 1:10
predicted <- 1.1 * y
murphy_diagram(y, predicted, theta = seq(0.9, 1.2, by = 0.01))
two_models <- cbind(m1 = predicted, m2 = 1.2 * y)
murphy_diagram(y, two_models, theta = seq(0.9, 1.3, by = 0.01))
```

---

performance

*Performance*

---

### Description

Applies one or more metrics to a data.frame containing columns with actual and predicted values as well as an optional column with case weights. The results are returned as a data.frame and can be used in a pipe.

### Usage

```
performance(
  data,
  actual,
  predicted,
  w = NULL,
  metrics = rmse,
```

```

    key = "metric",
    value = "value",
    ...
  )

```

### Arguments

<code>data</code>	A data frame with columns <code>actual</code> , <code>predicted</code> , and optionally <code>w</code> .
<code>actual</code>	The column name in <code>data</code> referring to actual values.
<code>predicted</code>	The column name in <code>data</code> referring to predicted values.
<code>w</code>	The optional column name in <code>data</code> referring to case weights.
<code>metrics</code>	Either a function or a named list of functions. Each function represents a metric and has four arguments: <ul style="list-style-type: none"> <li>• <code>observed</code>,</li> <li>• <code>predicted</code>,</li> <li>• <code>case weights</code>, and</li> <li>• <code>....</code></li> </ul> <p>If not a named list but a single function, the name of the function is guessed by <code>deparse(substitute(...))</code>, which would not provide the actual name of the function if called within <code>lapply()</code> etc. In such cases, you can pass a named list with one element, e.g., <code>list(rmse = rmse)</code>.</p>
<code>key</code>	Name of the resulting column containing the name of the metric. Defaults to "metric".
<code>value</code>	Name of the resulting column with the value of the metric. Defaults to "value".
<code>...</code>	Further arguments passed to the metric functions. E.g., if the metric is <code>r_squared()</code> , you could pass the relevant deviance function as additional argument (see examples).

### Value

Data frame with one row per metric and two columns: `key` and `value`.

### Examples

```

ir <- iris
fit_num <- lm(Sepal.Length ~ ., data = ir)
ir$fitted <- fit_num$fitted
performance(ir, "Sepal.Length", "fitted")
performance(ir, "Sepal.Length", "fitted", metrics = r_squared)
performance(
  ir,
  actual = "Sepal.Length",
  predicted = "fitted",
  metrics = c(`R-squared` = r_squared, rmse = rmse)
)
performance(
  ir,

```



```

    actual = "Sepal.Length",
    predicted = "fitted",
    metrics = r_squared,
    deviance_function = deviance_gamma
  )
performance(
  ir,
  actual = "Sepal.Length",
  predicted = "fitted",
  metrics = r_squared,
  deviance_function = deviance_tweedie,
  tweedie_p = 2
)

```

---

regression

*Regression Metrics*


---

## Description

Case-weighted versions of typical regression metrics:

- `mse()`: Mean-squared error.
- `rmse()`: Root-mean-squared error.
- `mae()`: Mean absolute error.
- `medae()`: Median absolute error.
- `mape()`: Mean absolute percentage error.
- `prop_within()`: Proportion of predictions that are within a given tolerance around the actual values.
- `deviance_normal()`: Average (unscaled) normal deviance. Equals MSE, and also the average Tweedie deviance with  $p = 0$ .
- `deviance_poisson()`: Average (unscaled) Poisson deviance. Equals average Tweedie deviance with  $p = 1$ .
- `deviance_gamma()`: Average (unscaled) Gamma deviance. Equals average Tweedie deviance with  $p = 2$ .
- `deviance_tweedie()`: Average Tweedie deviance with parameter  $p \in \{0\} \cup [1, \infty)$ , see reference.

Lower values mean better performance. Notable exception is `prop_within()`, where higher is better.

## Usage

```
mse(actual, predicted, w = NULL, ...)
```

```
rmse(actual, predicted, w = NULL, ...)
```

```
mae(actual, predicted, w = NULL, ...)  
medae(actual, predicted, w = NULL, ...)  
mape(actual, predicted, w = NULL, ...)  
prop_within(actual, predicted, w = NULL, tol = 1, ...)  
deviance_normal(actual, predicted, w = NULL, ...)  
deviance_poisson(actual, predicted, w = NULL, ...)  
deviance_gamma(actual, predicted, w = NULL, ...)  
deviance_tweedie(actual, predicted, w = NULL, tweedie_p = 0, ...)
```

### Arguments

actual	Observed values.
predicted	Predicted values.
w	Optional case weights.
...	Further arguments passed to <code>weighted_mean()</code> (no effect for <code>medae()</code> ).
tol	Predictions in $[\text{actual} \pm \text{tol}]$ count as "within" (only relevant for <code>prop_within()</code> ).
tweedie_p	Tweedie power $p \in \{0\} \cup [1, \infty)$ .

### Value

A numeric vector of length one.

### Input range

The values of `actual` and `predicted` can be any real numbers, with the following exceptions:

- `mape()`: Non-zero actual values.
- `deviance_poisson()`: Non-negative actual values, strictly positive predictions.
- `deviance_gamma()`: Strictly positive actual values and predictions.

### References

Jorgensen, B. (1997). The Theory of Dispersion Models. Chapman & Hall/CRC. ISBN 978-0412997112.

### Examples

```
y <- 1:10  
pred <- c(1:9, 12)  
w <- 1:10
```

```

rmse(y, pred)
sqrt(mse(y, pred)) # Same

mae(y, pred)
mae(y, pred, w = w)
medae(y, pred, w = 1:10)
mape(y, pred)

prop_within(y, pred)

deviance_normal(y, pred)
deviance_poisson(y, pred)
deviance_gamma(y, pred)

deviance_tweedie(y, pred, tweedie_p = 0) # Normal
deviance_tweedie(y, pred, tweedie_p = 1) # Poisson
deviance_tweedie(y, pred, tweedie_p = 2) # Gamma
deviance_tweedie(y, pred, tweedie_p = 1.5, w = 1:10)

```

---

rsquared

*Generalized R-Squared*


---

### Description

Returns (weighted) proportion of deviance explained, see reference below. For the mean-squared error as deviance, this equals the usual (weighted) R-squared. The higher, the better.

The convenience functions

- `r_squared_poisson()`,
- `r_squared_gamma()`, and
- `r_squared_bernoulli()`

call the function `r_squared(..., deviance_function = fun)` with the right deviance function.

### Usage

```

r_squared(
  actual,
  predicted,
  w = NULL,
  deviance_function = mse,
  reference_mean = NULL,
  ...
)

```

```
r_squared_poisson(actual, predicted, w = NULL, reference_mean = NULL, ...)
```

```
r_squared_gamma(actual, predicted, w = NULL, reference_mean = NULL, ...)
```

```
r_squared_bernoulli(actual, predicted, w = NULL, reference_mean = NULL, ...)
```

### Arguments

actual	Observed values.
predicted	Predicted values.
w	Optional case weights.
deviance_function	A positive (deviance) function taking four arguments: "actual", "predicted", "w" and "...". The default is <code>mse()</code> , which equals the average normal deviance.
reference_mean	An optional reference mean used to derive the null deviance. Recommended in out-of-sample applications.
...	Further arguments passed to <code>weighted_mean()</code> and <code>deviance_function()</code> .

### Details

The deviance gain is calculated regarding the null model derived from the actual values. While fine for in-sample considerations, this is only an approximation for out-of-sample considerations. There, it is recommended to calculate null deviance regarding the in-sample (weighted) mean. This value can be passed by the argument `reference_mean`.

### Value

A numeric vector of length one.

### References

Cohen, Jacob. et al. (2002). Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences (3rd ed.). Routledge. ISBN 978-0805822236.

### Examples

```
y <- 1:10
pred <- c(1, 1:9)
w <- 1:10

r_squared(y, pred)
r_squared(y, pred, w = w)

r_squared(y, pred, w = w, deviance_function = deviance_gamma)
r_squared_gamma(y, pred, w = w)

# Poisson situation
y2 <- 0:2
pred2 <- c(0.1, 1, 2)
r_squared(y2, pred2, deviance_function = deviance_poisson)
r_squared_poisson(y2, pred2)
```

```

# Binary (probabilistic) classification
y3 <- c(0, 0, 1, 1)
pred3 <- c(0.1, 0.1, 0.9, 0.8)
r_squared_bernoulli(y3, pred3, w = 1:4)

# With respect to 'own' deviance formula
myTweedie <- function(actual, predicted, w = NULL, ...) {
  deviance_tweedie(actual, predicted, w, tweedie_p = 1.5, ...)
}
r_squared(y, pred, deviance_function = myTweedie)

```

---

weighted\_cor

*Weighted Pearson Correlation*


---

### Description

Calculates weighted Pearson correlation coefficient between actual and predicted values by the help of `stats::cov.wt()`.

### Usage

```
weighted_cor(actual, predicted, w = NULL, na.rm = FALSE, ...)
```

### Arguments

actual	Observed values.
predicted	Predicted values.
w	Optional case weights.
na.rm	Should observations with missing values in actual or predicted be removed? Default is FALSE.
...	Further arguments passed to <code>stats::cov.wt()</code> .

### Value

A length-one numeric vector.

### Examples

```

weighted_cor(1:10, c(1, 1:9))
weighted_cor(1:10, c(1, 1:9), w = 1:10)

```

---

weighted_mean	<i>Weighted Mean</i>
---------------	----------------------

---

### Description

Returns the weighted mean of a numeric vector. In contrast to `stats::weighted.mean()`, `w` does not need to be specified.

### Usage

```
weighted_mean(x, w = NULL, ...)
```

### Arguments

<code>x</code>	Numeric vector.
<code>w</code>	Optional vector of non-negative case weights.
<code>...</code>	Further arguments passed to <code>mean()</code> or <code>stats::weighted.mean()</code> .

### Value

A length-one numeric vector.

### See Also

[stats::weighted.mean\(\)](#)

### Examples

```
weighted_mean(1:10)
weighted_mean(1:10, w = NULL)
weighted_mean(1:10, w = 1:10)
```

---

weighted_median	<i>Weighted Median</i>
-----------------	------------------------

---

### Description

Calculates weighted median based on [weighted\\_quantile\(\)](#).

### Usage

```
weighted_median(x, w = NULL, ...)
```

**Arguments**

`x` Numeric vector.  
`w` Optional vector of non-negative case weights.  
... Further arguments passed to `weighted_quantile()`.

**Value**

A length-one numeric vector.

**See Also**

[weighted\\_quantile\(\)](#)

**Examples**

```
n <- 21
x <- seq_len(n)
quantile(x, probs = 0.5)
weighted_median(x, w = rep(1, n))
weighted_median(x, w = x)
quantile(rep(x, x), probs = 0.5)
```

---

weighted\_quantile      *Weighted Quantiles*

---

**Description**

Calculates weighted quantiles based on the generalized inverse of the weighted ECDF. If no weights are passed, uses `stats::quantile()`.

**Usage**

```
weighted_quantile(
  x,
  w = NULL,
  probs = seq(0, 1, 0.25),
  na.rm = TRUE,
  names = TRUE,
  ...
)
```

**Arguments**

x	Numeric vector.
w	Optional vector of non-negative case weights.
probs	Vector of probabilities.
na.rm	Ignore missing data? Default is TRUE.
names	Return names? Default is TRUE.
...	Further arguments passed to <code>stats::quantile()</code> in the unweighted case. Not used in the weighted case.

**Value**

A length-one numeric vector.

**See Also**

[weighted\\_median\(\)](#)

**Examples**

```
n <- 10
x <- seq_len(n)
quantile(x)
weighted_quantile(x)
weighted_quantile(x, w = rep(1, n))
quantile(x, type = 1)
weighted_quantile(x, w = x) # same as Hmisc::wtd.quantile()
weighted_quantile(x, w = x, names = FALSE)
weighted_quantile(x, w = x, probs = 0.5, names = FALSE)

# Example with integer weights
x <- c(1, 1:11, 11, 11)
w <- seq_along(x)
weighted_quantile(x, w)
quantile(rep(x, w)) # same
```

---

weighted\_var

*Weighted Variance*

---

**Description**

Calculates weighted variance, see `stats::cov.wt()` or [https://en.wikipedia.org/wiki/Sample\\_mean\\_and\\_covariance#Weighted\\_samples](https://en.wikipedia.org/wiki/Sample_mean_and_covariance#Weighted_samples) for details.

**Usage**

```
weighted_var(x, w = NULL, method = c("unbiased", "ML"), na.rm = FALSE, ...)
```



**Arguments**

x	Numeric vector.
w	Optional vector of non-negative case weights.
method	Specifies how the result is scaled. If "unbiased", the denominator is reduced by 1, see <a href="#">stats::cov.wt()</a> for details.
na.rm	Should missing values in x be removed? Default is FALSE.
...	Further arguments passed to <a href="#">stats::cov.wt()</a> .

**Value**

A length-one numeric vector.

**See Also**

[stats::cov.wt\(\)](#)

**Examples**

```
weighted_var(1:10)
weighted_var(1:10, w = NULL)
weighted_var(1:10, w = rep(1, 10))
weighted_var(1:10, w = 1:10)
weighted_var(1:10, w = 1:10, method = "ML")
```

# Index

accuracy (classification), 2  
AUC (classification), 2  
  
classification, 2  
classification\_error (classification), 2  
  
deviance\_bernoulli (classification), 2  
deviance\_gamma (regression), 9  
deviance\_normal (regression), 9  
deviance\_poisson (regression), 9  
deviance\_tweedie (regression), 9  
  
elementary\_score, 4  
elementary\_score(), 7  
elementary\_score\_expectile  
    (elementary\_score), 4  
elementary\_score\_quantile  
    (elementary\_score), 4  
  
f1\_score (classification), 2  
  
gini\_coefficient (classification), 2  
graphics::matplot(), 7  
  
lapply(), 8  
logLoss (classification), 2  
  
mae (regression), 9  
mape (regression), 9  
mean(), 14  
medae (regression), 9  
mse (regression), 9  
multi\_metric, 5  
murphy\_diagram, 6  
murphy\_diagram(), 5  
  
performance, 7  
performance(), 6  
precision (classification), 2  
prop\_within (regression), 9  
  
r\_squared (rsquared), 11  
  
r\_squared(), 8  
r\_squared\_bernoulli (rsquared), 11  
r\_squared\_gamma (rsquared), 11  
r\_squared\_poisson (rsquared), 11  
recall (classification), 2  
regression, 9  
rmse (regression), 9  
rsquared, 11  
  
stats::cov.wt(), 13, 16, 17  
stats::quantile(), 15, 16  
stats::weighted.mean(), 14  
  
weighted\_cor, 13  
weighted\_mean, 14  
weighted\_mean(), 3, 5, 10, 12  
weighted\_median, 14  
weighted\_median(), 16  
weighted\_quantile, 15  
weighted\_quantile(), 14, 15  
weighted\_var, 16