Package ‘yardstick’

April 21, 2023

Type Package
Title Tidy Characterizations of Model Performance
Version 1.2.0
Description Tidy tools for quantifying how well model fits to a data set such as confusion matrices, class probability curve summaries, and regression metrics (e.g., RMSE).
License MIT + file LICENSE
BugReports https://github.com/tidymodels/yardstick/issues
Depends R (>= 3.4.0)
Imports cli, dplyr (>= 1.1.0), generics (>= 0.1.2), hardhat (>= 1.3.0), lifecycle (>= 1.0.3), rlang (>= 1.0.6), tibble, tidyselect (>= 1.2.0), utils, vctrs (>= 0.5.0)
Suggests covr, crayon, ggplot2, knitr, probably (>= 0.0.6), rmarkdown, survival (>= 3.5-0), testthat (>= 3.0.0), tidy
VignetteBuilder knitr
Config/Needs/website tidyverse/tidytemplate
Config/testthat/edition 3
Encoding UTF-8
LazyData true
RoxygenNote 7.2.3
NeedsCompilation yes
Author Max Kuhn [aut], Davis Vaughan [aut], Emil Hvitfeldt [aut, cre] (<https://orcid.org/0000-0002-0679-1945>), Posit Software, PBC [cph, fnd]
Maintainer Emil Hvitfeldt <emil.hvitfeldt@posit.co>
Repository CRAN
Date/Publication 2023-04-21 08:40:02 UTC
R topics documented:

accuracy .................................................. 3
average_precision ...................................... 4
bal_accuracy .............................................. 8
brier ..................................................... 10
ccc ....................................................... 12
check_metric ........................................... 15
classification_cost ..................................... 16
conf_mat .................................................. 19
detection_prevalence .................................. 22
developer-helpers ..................................... 24
f_meas .................................................... 27
gain_capture ........................................... 30
gain_curve .............................................. 34
hpc_cv .................................................... 37
huber_loss ............................................... 38
huber_loss_pseudo ....................................... 40
iic ....................................................... 42
j_index ................................................... 44
kap ....................................................... 47
lift_curve ............................................... 49
lung_surv ............................................... 52
mae .......................................................... 53
mape ..................................................... 55
mase ..................................................... 56
mcc ....................................................... 59
metric-summarizers .................................. 61
metrics .................................................. 64
metric_set ............................................... 66
metric_tweak ............................................ 68
mn_log_loss ............................................ 70
mpe ....................................................... 72
msd ....................................................... 74
new-metric ............................................. 76
npv ....................................................... 77
pathology ............................................... 81
poisson_log_loss ....................................... 81
ppv ....................................................... 83
precision ............................................... 86
pr_auc ................................................... 90
pr_curve ............................................... 93
recall ................................................... 95
rmse .................................................... 99
roc_auc .................................................. 101
roc_aunp ............................................... 104
roc_aunu ............................................... 107
roc_curve ................................................ 110
**Description**

Accuracy is the proportion of the data that are predicted correctly.

**Usage**

```r
accuracy(data, 

## S3 method for class 'data.frame'
accuracy(data, truth, estimate, na_rm = TRUE, case_weights = NULL, 

accuracy_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
```

**Arguments**

- **data**
  - Either a `data.frame` containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
  - `...`
    - Not currently used.

- **truth**
  - The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `vec()` functions, a factor vector.

- **estimate**
  - The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For `vec()` functions, a factor vector.

- **na_rm**
  - A logical value indicating whether NA values should be stripped before the computation proceeds.

- **case_weights**
  - The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For `vec()` functions, a numeric vector.
Value
A tibble with columns `.metric`, `.estimator`, and `.estimate` and 1 row of values.
For grouped data frames, the number of rows returned will be the same as the number of groups.
For `accuracy_vec()`, a single numeric value (or NA).

Multiclass
Accuracy extends naturally to multiclass scenarios. Because of this, macro and micro averaging are not implemented.

Author(s)
Max Kuhn

See Also
Other class metrics: `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `j_index()`, `kap()`, `mcc()`, `npv()`, `ppv()`, `precision()`, `recall()`, `sens()`, `spec()`

Examples
```r
library(dplyr)
data("two_class_example")
data("hpc_cv")

# Two class
accuracy(two_class_example, truth, predicted)

# Multiclass
# accuracy() has a natural multiclass extension
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  accuracy(obs, pred)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  accuracy(obs, pred)
```

---

average_precision  
*Area under the precision recall curve*

Description

`average_precision()` is an alternative to `pr_auc()` that avoids any ambiguity about what the value of precision should be when recall == 0 and there are not yet any false positive values (some say it should be 0, others say 1, others say undefined).

It computes a weighted average of the precision values returned from `pr_curve()`, where the weights are the increase in recall from the previous threshold. See `pr_curve()` for the full curve.
average_precision

Usage

average_precision(data, ...)

## S3 method for class 'data.frame'

average_precision(
    data,
    truth,
    ..., 
    estimator = NULL,
    na_rm = TRUE,
    event_level = yardstick_event_level(),
    case_weights = NULL
)

average_precision_vec(
    truth,
    estimate,
    estimator = NULL,
    na_rm = TRUE,
    event_level = yardstick_event_level(),
    case_weights = NULL,
    ...
)

Arguments

data       A data.frame containing the columns specified by truth and ... .
...         A set of unquoted column names or one or more dplyr selector functions to
            choose which variables contain the class probabilities. If truth is binary, only
            1 column should be selected, and it should correspond to the value of event_level.
            Otherwise, there should be as many columns as factor levels of truth and the
            ordering of the columns should be the same as the factor levels of truth.

truth      The column identifier for the true class results (that is a factor). This should be
            an unquoted column name although this argument is passed by expression and
            supports quasiquotation (you can unquote column names). For _vec() func-
            tions, a factor vector.

estimator  One of “binary”, “macro”, or “macro_weighted” to specify the type of aver-
            aging to be done. “binary” is only relevant for the two class case. The other
            two are general methods for calculating multiclass metrics. The default will
            automatically choose “binary” or “macro” based on truth.

na_rm      A logical value indicating whether NA values should be stripped before the
            computation proceeds.

event_level A single string. Either “first” or “second” to specify which level of truth
            to consider as the “event”. This argument is only applicable when estimator =
            “binary”. The default uses an internal helper that generally defaults to “first”,
            however, if the deprecated global option yardstick.event_first is set, that
            will be used instead with a warning.
average_precision

case_weights  The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

estimate  If truth is binary, a numeric vector of class probabilities corresponding to the "relevant" class. Otherwise, a matrix with as many columns as factor levels of truth. It is assumed that these are in the same order as the levels of truth.

Details

The computation for average precision is a weighted average of the precision values. Assuming you have n rows returned from pr_curve(), it is a sum from 2 to n, multiplying the precision value \( p_i \) by the increase in recall over the previous threshold, \( r_i - r_{i-1} \).

\[
AP = \sum (r_i - r_{i-1}) \times p_i
\]

By summing from 2 to n, the precision value \( p_1 \) is never used. While pr_curve() returns a value for \( p_1 \), it is technically undefined as \( tp / (tp + fp) \) with \( tp = 0 \) and \( fp = 0 \). A common convention is to use 1 for \( p_1 \), but this metric has the nice property of avoiding the ambiguity. On the other hand, \( r_1 \) is well defined as long as there are some events (p), and it is \( tp / p \) with \( tp = 0 \), so \( r_1 = 0 \).

When \( p_1 \) is defined as 1, the average_precision() and roc_auc() values are often very close to one another.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For average_precision_vec(), a single numeric value (or NA).

Multiclass

Macro and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.
See Also

pr_curve() for computing the full precision recall curve.
pr_auc() for computing the area under the precision recall curve using the trapezoidal rule.

Other class probability metrics: brier_class(), classification_cost(), gain_capture(), mn_log_loss(), pr_auc(), roc_auc(), roc_aunp(), roc_aunu()

Examples

```r
# Two class example

data(two_class_example)

# Binary metrics using class probabilities take a factor 'truth' column, # and a single class probability column containing the probabilities of # the event of interest. Here, since "Class1" is the first level of # "truth", it is the event of interest and we pass in probabilities for it. average_precision(two_class_example, truth, Class1)

# Multiclass example

data(hpc_cv)

# You can use the col1:colN tidyselect syntax
library(dplyr)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
  average_precision(obs, VF:L)

# Change the first level of obs from "VF" to "M" to alter the # event of interest. The class probability columns should be supplied # in the same order as the levels.

hpc_cv %>%
  filter(Resample == "Fold01") %>%
  mutate(obs = relevel(obs, "M")) %>%
  average_precision(obs, M, VF:L)

# Groups are respected

hpc_cv %>%
  group_by(Resample) %>%
  average_precision(obs, VF:L)

# Weighted macro averaging

hpc_cv %>%
```

average_precision

---

**average_precision**

See Also

- pr_curve()
- pr_auc()

Other class probability metrics: brier_class(), classification_cost(), gain_capture(), mn_log_loss(), pr_auc(), roc_auc(), roc_aunp(), roc_aunu()
```r
group_by(Resample) %>%
  average_precision(obs, VF:L, estimator = "macro_weighted")

# Vector version
# Supply a matrix of class probabilities
fold1 <- hpc_cv %>%
  filter(Resample == "Fold01")

average_precision_vec(
  truth = fold1$obs,
  matrix(
    c(fold1$VF, fold1$F, fold1$M, fold1$L),
    ncol = 4
  )
)
```

---

### bal_accuracy

**Balanced accuracy**

**Description**

Balanced accuracy is computed here as the average of `sens()` and `spec()`.

**Usage**

```r
bal_accuracy(data, ...)
```

```r
## S3 method for class 'data.frame'
bal_accuracy(
  data,
  truth,
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)
```

```r
bal_accuracy_vec(
  truth,
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)
```
Arguments

data Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.

... Not currently used.

truth The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

estimate The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

estimator One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

na.rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

event_level A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For bal_accuracy_vec(), a single numeric value (or NA).

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

Macro, micro, and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.
Author(s)
Max Kuhn

See Also
Other class metrics: `accuracy()`, `detection_prevalence()`, `f_meas()`, `j_index()`, `kap()`, `mcc()`, `npv()`, `ppv()`, `precision()`, `recall()`, `sens()`, `spec()`

Examples

```r
# Two class
data("two_class_example")
bal_accuracy(two_class_example, truth, predicted)

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
bal_accuracy(obs, pred)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
bal_accuracy(obs, pred)

# Weighted macro averaging
hpc_cv %>%
  group_by(Resample) %>%
bal_accuracy(obs, pred, estimator = "macro_weighted")

# Vector version
bal_accuracy_vec(
  two_class_example$truth,
  two_class_example$predicted
)

# Making Class2 the "relevant" level
bal_accuracy_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)
```

---

brier_class  

*Brier score for classification models*
**brier_class**

**Description**
Compute the Brier score for a classification model.

**Usage**

```r
brier_class(data, ...)  
## S3 method for class 'data.frame'

brier_class(data, truth, ..., na_rm = TRUE, case_weights = NULL)

brier_class_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
```

**Arguments**

- `data` (A data.frame containing the columns specified by truth and ...)
- `truth` (The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.)
- `na_rm` (A logical value indicating whether NA values should be stripped before the computation proceeds.)
- `case_weights` (The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.)
- `estimate` (If truth is binary, a numeric vector of class probabilities corresponding to the "relevant" class. Otherwise, a matrix with as many columns as factor levels of truth. It is assumed that these are in the same order as the levels of truth.)

**Details**

The Brier score is analogous to the mean squared error in regression models. The difference between a binary indicator for a class and its corresponding class probability are squared and averaged.

This function uses the convention in Kruppa et al (2014) and divides the result by two.

Smaller values of the score are associated with better model performance.

**Value**

A tibble with columns _.metric_. _, estimator_, and _.estimate_ and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For brier_class_vec(), a single numeric value (or NA).
Multiclass

Brier scores can be computed in the same way for any number of classes. Because of this, no averaging types are supported.

Author(s)

Max Kuhn

References


See Also

Other class probability metrics: `average_precision()`, `classification_cost()`, `gain_capture()`, `mn_log_loss()`, `pr_auc()`, `roc_auc()`, `roc_aunp()`, `roc_aunu()`

Examples

```r
# Two class
data("two_class_example")
brier_class(two_class_example, truth, Class1)

# Multiclass
library(dplyr)
data(hpc_cv)

# You can use the col1:colN tidyselect syntax
hpc_cv %>%
  filter(Resample == "Fold01") %>%
brier_class(obs, VF:L)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
brier_class(obs, VF:L)
```

---

**Description**

Calculate the concordance correlation coefficient.
Usage

```r
ccc(data, ...)
```

```r
## S3 method for class 'data.frame'
ccc(
  data,
  truth,
  estimate,
  bias = FALSE,
  na_rm = TRUE,
  case_weights = NULL,
  ...
)
```

```r
ccc_vec(truth, estimate, bias = FALSE, na_rm = TRUE, case_weights = NULL, ...)
```

Arguments

- `data` A `data.frame` containing the columns specified by the `truth` and `estimate` arguments.
- `...` Not currently used.
- `truth` The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a numeric vector.
- `estimate` The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a numeric vector.
- `bias` A logical; should the biased estimate of variance be used (as is Lin (1989))? 
- `na_rm` A logical value indicating whether NA values should be stripped before the computation proceeds.
- `case_weights` The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector.

Details

`ccc()` is a metric of both consistency/correlation and accuracy, while metrics such as `rmse()` are strictly for accuracy and metrics such as `rsq()` are strictly for consistency/correlation.

Value

A `tibble` with columns `.metric`, `.estimator`, and `.estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For `ccc_vec()`, a single numeric value (or NA).
Author(s)
Max Kuhn

References

See Also
Other numeric metrics: `huber_loss_pseudo()`, `huber_loss()`, `iic()`, `mae()`, `mape()`, `mase()`, `mpe()`, `msd()`, `poisson_log_loss()`, `rmse()`, `rpd()`, `rpiq()`, `rsq_trad()`, `rsq()`, `smape()`
Other consistency metrics: `rpd()`, `rpiq()`, `rsq_trad()`, `rsq()`
Other accuracy metrics: `huber_loss_pseudo()`, `huber_loss()`, `iic()`, `mae()`, `mape()`, `mase()`, `mpe()`, `msd()`, `poisson_log_loss()`, `rmse()`, `smape()`

Examples
# Supply truth and predictions as bare column names
ccc(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample)
  ccc(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))
check_metric

Developer function for checking inputs in new metrics

Description

check_numeric_metric(), check_class_metric(), and check_prob_metric() are useful alongside metric-summarizers for implementing new custom metrics. metric-summarizers call the metric function inside dplyr::summarise(). These functions perform checks on the inputs in accordance with the type of metric that is used.

Usage

check_numeric_metric(truth, estimate, case_weights, call = caller_env())

check_class_metric(
  truth,
  estimate,
  case_weights,
  estimator,
  call = caller_env()
)

check_prob_metric(
  truth,
  estimate,
  case_weights,
  estimator,
  call = caller_env()
)

check_dynamic_survival_metric(
  truth,
  estimate,
  case_weights,
  call = caller_env()
)

check_static_survival_metric(
  truth,
  estimate,
  case_weights,
  call = caller_env()
)

Arguments

truth The realized vector of truth.
classification_cost

Costs function for poor classification

Description

classification_cost() calculates the cost of a poor prediction based on user-defined costs. The costs are multiplied by the estimated class probabilities and the mean cost is returned.

Usage

classification_cost(data, ...)

# S3 method for class 'data.frame'
classification_cost(
  data,
  truth,
  ..., 
  costs = NULL,
  na_rm = TRUE, 
)
classification_cost

event_level = yardstick_event_level(),
case_weights = NULL
)

classification_cost_vec(
  truth,
estimate,
costs = NULL,
na_rm = TRUE,
event_level = yardstick_event_level(),
case_weights = NULL,
...
)

Arguments

data  A data.frame containing the columns specified by truth and ...  

...  A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.

truth  The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

costs  A data frame with columns "truth", "estimate", and "cost". "truth" and "estimate" should be character columns containing unique combinations of the levels of the truth factor. "costs" should be a numeric column representing the cost that should be applied when the "estimate" is predicted, but the true result is "truth". It is often the case that when "truth" == "estimate", the cost is zero (no penalty for correct predictions). If any combinations of the levels of truth are missing, their costs are assumed to be zero. If NULL, equal costs are used, applying a cost of 0 to correct predictions, and a cost of 1 to incorrect predictions.

na_rm  A logical value indicating whether NA values should be stripped before the computation proceeds.

event_level  A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

case_weights  The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.
estimate

If truth is binary, a numeric vector of class probabilities corresponding to the "relevant" class. Otherwise, a matrix with as many columns as factor levels of truth. It is assumed that these are in the same order as the levels of truth.

Details

As an example, suppose that there are three classes: "A", "B", and "C". Suppose there is a truly "A" observation with class probabilities $A = 0.3 / B = 0.3 / C = 0.4$. Suppose that, when the true result is class "A", the costs for each class were $A = 0 / B = 5 / C = 10$, penalizing the probability of incorrectly predicting "C" more than predicting "B". The cost for this prediction would be $0.3 \times 0 + 0.3 \times 5 + 0.4 \times 10$. This calculation is done for each sample and the individual costs are averaged.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For class_cost_vec(), a single numeric value (or NA).

Author(s)

Max Kuhn

See Also

Other class probability metrics: average_precision(), brier_class(), gain_capture(), mn_log_loss(), pr_auc(), roc_auc(), roc_aunp(), roc_aunu()

Examples

library(dplyr)

# Two class example
data(two_class_example)

data(two_class_example)

classification_cost(two_class_example, truth, Class1, costs = costs1)
### Description

Calculates a cross-tabulation of observed and predicted classes.

### Usage

```r
classification_cost(two_class_example, truth, Class1, costs = costs2)
```

```r
# Multiclass
data(hpc_cv)

# Define cost matrix from Kuhn and Johnson (2013)
hpc_costs <- tribble(
  ~estimate, ~truth, ~cost,
  "VF", "VF", 0,
  "VF", "F", 1,
  "VF", "M", 5,
  "VF", "L", 10,
  "F", "VF", 1,
  "F", "F", 0,
  "F", "M", 5,
  "F", "L", 5,
  "M", "VF", 1,
  "M", "F", 1,
  "M", "M", 0,
  "M", "L", 1,
  "L", "VF", 1,
  "L", "F", 1,
  "L", "M", 1,
  "L", "L", 0
)

# You can use the col1:colN tidyselect syntax
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  classification_cost(obs, VF:L, costs = hpc_costs)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  classification_cost(obs, VF:L, costs = hpc_costs)
```

---

**conf_mat**

Confusion Matrix for Categorical Data

---

**Description**

Calculates a cross-tabulation of observed and predicted classes.

**Usage**

```r
classification_cost(two_class_example, truth, Class1, costs = costs2)
```

```r
# Multiclass
data(hpc_cv)

# Define cost matrix from Kuhn and Johnson (2013)
hpc_costs <- tribble(
  ~estimate, ~truth, ~cost,
  "VF", "VF", 0,
  "VF", "F", 1,
  "VF", "M", 5,
  "VF", "L", 10,
  "F", "VF", 1,
  "F", "F", 0,
  "F", "M", 5,
  "F", "L", 5,
  "M", "VF", 1,
  "M", "F", 1,
  "M", "M", 0,
  "M", "L", 1,
  "L", "VF", 1,
  "L", "F", 1,
  "L", "M", 1,
  "L", "L", 0
)

# You can use the col1:colN tidyselect syntax
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  classification_cost(obs, VF:L, costs = hpc_costs)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  classification_cost(obs, VF:L, costs = hpc_costs)
```
data,  
truth,  
estimate,  
dnn = c("Prediction", "Truth"),  
...  
)

## S3 method for class 'conf_mat'
tidy(x, ...)

### Arguments

- **data**: A data frame or a `base::table()`.
- **...**: Not used.
- **truth**: The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a factor vector.
- **estimate**: The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a factor vector.
- **dnn**: A character vector of dimnames for the table.
- **case_weights**: The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector.
- **x**: A `conf_mat` object.

### Details

For `conf_mat()` objects, a broom tidy() method has been created that collapses the cell counts by cell into a data frame for easy manipulation.

There is also a `summary()` method that computes various classification metrics at once. See `summary.conf_mat()`

There is a `ggplot2::autoplot()` method for quickly visualizing the matrix. Both a heatmap and mosaic type is implemented.

The function requires that the factors have exactly the same levels.

### Value

- `conf_mat()` produces an object with class `conf_mat`. This contains the table and other objects.
- `tidy.conf_mat()` generates a tibble with columns `name` (the cell identifier) and `value` (the cell count).

When used on a grouped data frame, `conf_mat()` returns a tibble containing columns for the groups along with `conf_mat`, a list-column where each element is a `conf_mat` object.
See Also

`summary.conf_mat()` for computing a large number of metrics from one confusion matrix.

Examples

```r
library(dplyr)
data("hpc_cv")

# The confusion matrix from a single assessment set (i.e. fold)
cm <- hpc_cv %>%
  filter(Resample == "Fold01") %>%
  conf_mat(obs, pred)
cm

# Now compute the average confusion matrix across all folds in
# terms of the proportion of the data contained in each cell.
# First get the raw cell counts per fold using the `tidy` method
library(tidyr)

cells_per_resample <- hpc_cv %>%
  group_by(Resample) %>%
  conf_mat(obs, pred) %>%
  mutate(tidied = lapply(conf_mat, tidy)) %>%
  unnest(tidied)

# Get the totals per resample
counts_per_resample <- hpc_cv %>%
  group_by(Resample) %>%
  summarize(total = n()) %>%
  left_join(cells_per_resample, by = "Resample") %>%
  # Compute the proportions
  mutate(prop = value / total) %>%
  group_by(name) %>%
  # Average
  summarize(prop = mean(prop))

counts_per_resample

# Now reshape these into a matrix
mean_cmat <- matrix(counts_per_resample$prop, byrow = TRUE, ncol = 4)
rownames(mean_cmat) <- levels(hpc_cv$obs)
colnames(mean_cmat) <- levels(hpc_cv$obs)

round(mean_cmat, 3)

# The confusion matrix can quickly be visualized using autoplot()
library(ggplot2)

autoplot(cm, type = "mosaic")
autoplot(cm, type = "heatmap")
```
detection_prevalence  

Description

Detection prevalence is defined as the number of predicted positive events (both true positive and false positive) divided by the total number of predictions.

Usage

detection_prevalence(data, ...)

## S3 method for class 'data.frame'
detection_prevalence(  
data,  
truth,  
estimate,  
estimator = NULL,  
na.rm = TRUE,  
case_weights = NULL,  
event_level = yardstick_event_level(),  
...  
)

detection_prevalence_vec(  
truth,  
estimate,  
estimator = NULL,  
na.rm = TRUE,  
case_weights = NULL,  
event_level = yardstick_event_level(),  
...  
)

Arguments

data Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.

... Not currently used.

truth The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

estimate The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
detection_prevalence

estimator One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

event_level A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For detection_prevalence_vec(), a single numeric value (or NA).

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

Macro, micro, and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.

Author(s)

Max Kuhn

See Also

Other class metrics: accuracy(), bal_accuracy(), f_meas(), j_index(), kap(), mcc(), npv(), ppv(), precision(), recall(), sens(), spec()
Examples

```r
# Two class
data("two_class_example")
detection_prevalence(two_class_example, truth, predicted)

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
detection_prevalence(obs, pred)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
detection_prevalence(obs, pred)

# Weighted macro averaging
hpc_cv %>%
  group_by(Resample) %>%
detection_prevalence(obs, pred, estimator = "macro_weighted")

# Vector version
detection_prevalence_vec(
  two_class_example$truth,
  two_class_example$predicted
)

# Making Class2 the "relevant" level
detection_prevalence_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)
```

---

**developer-helpers**  
**Developer helpers**

**Description**

Helpers to be used alongside check_metric, yardstick_remove_missing and metric summarizers when creating new metrics. See Custom performance metrics for more information.

**Usage**

```r
dots_to_estimate(data, ...)

get_weights(data, estimator)
```
finalize_estimator(  
  x,  
estimator = NULL,  
metric_class = "default",  
call = caller_env()  
)

finalize_estimator_internal(  
metric_dispatcher,  
x,  
estimator,  
call = caller_env()  
)

validate_estimator(estimator, estimator_override = NULL, call = caller_env())

Arguments

data
A table with truth values as columns and predicted values as rows.

...  
A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.

estimator  
Either NULL for auto-selection, or a single character for the type of estimator to use.

x
The column used to autoselect the estimator. This is generally the truth column, but can also be a table if your metric has table methods.

metric_class
A single character of the name of the metric to autoselect the estimator for. This should match the method name created for finalize_estimator_internal().

call
The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.

metric_dispatcher
A simple dummy object with the class provided to metric_class. This is created and passed along for you.

estimator_override
A character vector overriding the default allowed estimator list of c("binary", "macro", "micro", "macro_weighted"). Set this if your classification estimator does not support all of these methods.

Dots -> Estimate

[Deprecated]

dots_to_estimate() is useful with class probability metrics that take ... rather than estimate as an argument. It constructs either a single name if 1 input is provided to ... or it constructs a
quosure where the expression constructs a matrix of as many columns as are provided to . . . . These are eventually evaluated in the `summarise()` call in `metric-summarizers` and evaluate to either a vector or a matrix for further use in the underlying vector functions.

**Weight Calculation**

get_weights() accepts a confusion matrix and an estimator of type "macro", "micro", or "macro_weighted" and returns the correct weights. It is useful when creating multiclass metrics.

**Estimator Selection**

finalize_estimator() is the engine for auto-selection of estimator based on the type of x. Generally x is the truth column. This function is called from the vector method of your metric.

finalize_estimator_internal() is an S3 generic that you should extend for your metric if it does not implement only the following estimator types: "binary", "macro", "micro", and "macro_weighted". If your metric does support all of these, the default version of finalize_estimator_internal() will autoselect estimator appropriately. If you need to create a method, it should take the form: finalize_estimator_internal(metric_name). Your method for finalize_estimator_internal() should do two things:

1. If estimator is NULL, autoselect the estimator based on the type of x and return a single character for the estimator.
2. If estimator is not NULL, validate that it is an allowed estimator for your metric and return it.

If you are using the default for finalize_estimator_internal(), the estimator is selected using the following heuristics:

1. If estimator is not NULL, it is validated and returned immediately as no auto-selection is needed.
2. If x is a:
   - factor - Then "binary" is returned if it has 2 levels, otherwise "macro" is returned.
   - numeric - Then "binary" is returned.
   - table - Then "binary" is returned if it has 2 columns, otherwise "macro" is returned.
     This is useful if you have table methods.
   - matrix - Then "macro" is returned.

**Estimator Validation**

validate_estimator() is called from your metric specific method of finalize_estimator_internal() and ensures that a user provided estimator is of the right format and is one of the allowed values.

**See Also**

`metric-summarizers` `check_metric` `yardstick_remove_missing`
Description

These functions calculate the $f_{\text{meas}}()$ of a measurement system for finding relevant documents compared to reference results (the truth regarding relevance). Highly related functions are $\text{recall}()$ and $\text{precision}()$.

Usage

\begin{verbatim}
f_meas(data, ...)  
## S3 method for class 'data.frame'
f_meas(
  data,
  truth,
  estimate,
  beta = 1,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

f_meas_vec(
  truth,
  estimate,
  beta = 1,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)
\end{verbatim}

Arguments

- **data**: Either a `data.frame` containing the columns specified by the `truth` and `estimate` arguments, or a table/matrix where the true class results should be in the columns of the table.
- **...**: Not currently used.
- **truth**: The column identifier for the true class results (that is a `factor`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `factor` vector.
**estimate**

The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

**beta**

A numeric value used to weight precision and recall. A value of 1 is traditionally used and corresponds to the harmonic mean of the two values but other values weight recall beta times more important than precision.

**estimator**

One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

**na_rm**

A logical value indicating whether NA values should be stripped before the computation proceeds.

**case_weights**

The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

**event_level**

A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

**Details**

The measure "F" is a combination of precision and recall (see below).

**Value**

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For f_meas_vec(), a single numeric value (or NA).

**Relevant Level**

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

**Multiclass**

Macro, micro, and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.
**Implementation**

Suppose a 2x2 table with notation:

<table>
<thead>
<tr>
<th></th>
<th>Reference</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Irrelevant</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

The formulas used here are:

\[
\text{recall} = \frac{A}{A + C} \\
\text{precision} = \frac{A}{A + B} \\
F_{\text{meas}} = \frac{1 + \beta^2 \cdot \text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}
\]

See the references for discussions of the statistics.

**Author(s)**

Max Kuhn

**References**


**See Also**

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `j_index()`, `kap()`, `mcc()`, `npv()`, `ppv()`, `precision()`, `recall()`, `sens()`, `spec()`

Other relevance metrics: `precision()`, `recall()`

**Examples**

```r
# Two class
data("two_class_example")
f_meas(two_class_example, truth, predicted)

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv %>% filter(Resample == "Fold01") %>%
f_meas(obs, pred)
```
# Groups are respected
hpc_cv %>%
group_by(Resample) %>%
f_meas(obs, pred)

# Weighted macro averaging
hpc_cv %>%
group_by(Resample) %>%
f_meas(obs, pred, estimator = "macro_weighted")

# Vector version
f_meas_vec(
  two_class_example$truth,
  two_class_example$predicted
)

# Making Class2 the "relevant" level
f_meas_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)

gain_capture

Gain capture

Description

gain_capture() is a measure of performance similar to an AUC calculation, but applied to a gain curve.

Usage

gain_capture(data, ...)

## S3 method for class 'data.frame'
gain_capture(
  data,
  truth,
  ..., 
estimator = NULL,
  na.rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL
)

gain_capture_vec(
  truth,
  estimate,


\begin{verbatim}
  estimator = NULL,
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL,
  ...
)

Arguments

  data    A data.frame containing the columns specified by truth and ....

  ...    A set of unquoted column names or one or more dplyr selector functions to
          choose which variables contain the class probabilities. If truth is binary, only 1
          column should be selected, and it should correspond to the value of event_level.
          Otherwise, there should be as many columns as factor levels of truth and the
          ordering of the columns should be the same as the factor levels of truth.

  truth   The column identifier for the true class results (that is a factor). This should
          be an unquoted column name although this argument is passed by expression and
          supports quasiquotation (you can unquote column names). For _vec() functions, a
          factor vector.

  estimator    One of "binary", "macro", or "macro_weighted" to specify the type of aver-
               aging to be done. "binary" is only relevant for the two class case. The other
               two are general methods for calculating multiclass metrics. The default will
               automatically choose "binary" or "macro" based on truth.

  na_rm    A logical value indicating whether NA values should be stripped before the
           computation proceeds.

  event_level    A single string. Either "first" or "second" to specify which level of truth
                 to consider as the "event". This argument is only applicable when estimator =
                 "binary". The default uses an internal helper that generally defaults to "first",
                 however, if the deprecated global option yardstick.event_first is set, that
                 will be used instead with a warning.

  case_weights    The optional column identifier for case weights. This should be an unquoted
                 column name that evaluates to a numeric column in data. For _vec() functions,
                 a numeric vector.

  estimate    If truth is binary, a numeric vector of class probabilities corresponding to the
               "relevant" class. Otherwise, a matrix with as many columns as factor levels of
               truth. \textit{It is assumed that these are in the same order as the levels of truth.}

Details

  gain_capture() calculates the area \textit{under} the gain curve, but \textit{above} the baseline, and then divides
  that by the area \textit{under} a perfect gain curve, but \textit{above} the baseline. It is meant to represent the
  amount of potential gain "captured" by the model.

  The \texttt{gain_capture()} metric is identical to the accuracy ratio (AR), which is also sometimes called
  the gini coefficient. These two are generally calculated on a cumulative accuracy profile curve, but
  this is the same as a gain curve. See the Engelmann reference for more information.
\end{verbatim}
Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For gain_capture_vec(), a single numeric value (or NA).

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

Macro and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.

Author(s)

Max Kuhn

References


See Also

gain_curve() to compute the full gain curve.

Other class probability metrics: average_precision(), brier_class(), classification_cost(), mn_log_loss(), pr_auc(), roc_auc(), roc_aunp(), roc_aunu()

Examples

# Two class example

# `truth` is a 2 level factor. The first level is "Class1", which is the # "event of interest" by default in yardstick. See the Relevant Level # section above.
# data(two_class_example)

# Binary metrics using class probabilities take a factor `truth` column, # and a single class probability column containing the probabilities of # the event of interest. Here, since "Class1" is the first level of
# "truth", it is the event of interest and we pass in probabilities for it.
gain_capture(two_class_example, truth, Class1)

# Multiclass example

# `obs` is a 4 level factor. The first level is "VF", which is the
# "event of interest" by default in yardstick. See the Relevant Level
# section above.
data(hpc_cv)

# You can use the col1:colN tidyselect syntax
library(dplyr)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
  gain_capture(obs, VF:L)

# Change the first level of `obs` from "VF" to "M" to alter the
# event of interest. The class probability columns should be supplied
# in the same order as the levels.

hpc_cv %>%
  filter(Resample == "Fold01") %>%
  mutate(obs = relevel(obs, "M")) %>%
  gain_capture(obs, M, VF:L)

# Groups are respected

hpc_cv %>%
  group_by(Resample) %>%
  gain_capture(obs, VF:L)

# Weighted macro averaging

hpc_cv %>%
  group_by(Resample) %>%
  gain_capture(obs, VF:L, estimator = "macro_weighted")

# Vector version

# Supply a matrix of class probabilities
fold1 <- hpc_cv %>

filter(Resample == "Fold01")

gain_capture_vec(
  truth = fold1$obs,
  matrix(  
c(fold1$VF, fold1$F, fold1$M, fold1$L),
  ncol = 4
  )
)

# Visually, this represents the area under the black curve, but above the
# 45 degree line, divided by the area of the shaded triangle.
library(ggplot2)
autoplot(gain_curve(two_class_example, truth, Class1))

### Description

gain_curve() constructs the full gain curve and returns a tibble. See gain_capture() for the relevant area under the gain curve. Also see lift_curve() for a closely related concept.

### Usage

gain_curve(data, ...)

```r
## S3 method for class 'data.frame'
gain_curve(
  data,
  truth,
  ..., 
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL
)
```

### Arguments

data A data.frame containing the columns specified by truth and ....

... A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.

truth The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

event_level A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.
Details

There is a `ggplot2::autoplot()` method for quickly visualizing the curve. This works for binary and multiclass output, and also works with grouped data (i.e. from resamples). See the examples.

The greater the area between the gain curve and the baseline, the better the model.

Gain curves are identical to CAP curves (cumulative accuracy profile). See the Engelmann reference for more information on CAP curves.

Value

A tibble with class `gain_df` or `gain_grouped_df` having columns:

- `.n` The index of the current sample.
- `.n_events` The index of the current unique sample. Values with repeated `estimate` values are given identical indices in this column.
- `.percent_tested` The cumulative percentage of values tested.
- `.percent_found` The cumulative percentage of true results relative to the total number of true results.

If using the `case_weights` argument, all of the above columns will be weighted. This makes the most sense with frequency weights, which are integer weights representing the number of times a particular observation should be repeated.

Gain and Lift Curves

The motivation behind cumulative gain and lift charts is as a visual method to determine the effectiveness of a model when compared to the results one might expect without a model. As an example, without a model, if you were to advertise to a random 10% of your customer base, then you might expect to capture 10% of the of the total number of positive responses had you advertised to your entire customer base. Given a model that predicts which customers are more likely to respond, the hope is that you can more accurately target 10% of your customer base and capture >10% of the total number of positive responses.

The calculation to construct gain curves is as follows:

1. `truth` and `estimate` are placed in descending order by the `estimate` values (`estimate` here is a single column supplied in . . .).
2. The cumulative number of samples with true results relative to the entire number of true results are found. This is the y-axis in a gain chart.

Multiclass

If a multiclass `truth` column is provided, a one-vs-all approach will be taken to calculate multiple curves, one per level. In this case, there will be an additional column, `.level1`, identifying the "one" column in the one-vs-all calculation.
Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Author(s)

Max Kuhn

References


See Also

Compute the relevant area under the gain curve with gain_capture(). Other curve metrics: lift_curve(), pr_curve(), roc_curve()

Examples

```r
# Two class example
# `truth` is a 2 level factor. The first level is "Class1", which is the # "event of interest" by default in yardstick. See the Relevant Level # section above.
# Binary metrics using class probabilities take a factor `truth` column, # and a single class probability column containing the probabilities of # the event of interest. Here, since "Class1" is the first level of # "truth", it is the event of interest and we pass in probabilities for it.
gain_curve(two_class_example, truth, Class1)
```

```r
# `autoplot()
library(ggplot2)
library(dplyr)

# Use autoplot to visualize
# The top left hand corner of the grey triangle is a "perfect" gain curve
autoplot(gain_curve(two_class_example, truth, Class1))
```

# Multiclass one-vs-all approach
# One curve per level
hpc_cv](#) %>%
  filter(Resample == "Fold01") %>%
  gain_curve(obs, VF:L) %>%
  autoplot()

# Same as above, but will all of the resamples
# The resample with the minimum (farthest to the left) "perfect" value is
# used to draw the shaded region
hpc_cv %>%
  group_by(Resample) %>%
  gain_curve(obs, VF:L) %>%
  autoplot()

---

### Description

Multiclass Probability Predictions

### Details

This data frame contains the predicted classes and class probabilities for a linear discriminant analysis model fit to the HPC data set from Kuhn and Johnson (2013). These data are the assessment sets from a 10-fold cross-validation scheme. The data column columns for the true class (`obs`), the class prediction (`pred`) and columns for each class probability (columns VF, F, M, and L). Additionally, a column for the resample indicator is included.

### Value

- `hpc_cv`: a data frame

### Source


### Examples

data(hpc_cv)
str(hpc_cv)

# `obs` is a 4 level factor. The first level is "VF", which is the
# "event of interest" by default in yardstick. See the Relevant Level
# section in any classification function (such as `pr_auc`) to see how
# to change this.
levels(hpc_cv$obs)
huber_loss  

Description

Calculate the Huber loss, a loss function used in robust regression. This loss function is less sensitive to outliers than \texttt{rmse\texttt{}}. This function is quadratic for small residual values and linear for large residual values.

Usage

\begin{verbatim}
huber_loss(data, ...)  

## S3 method for class 'data.frame'
huber_loss(  
data,  
truth,  
estimate,  
\text{delta} = 1,  
\text{na_rm} = \text{TRUE},  
\text{case_weights} = \text{NULL},  
\ldots  
)

huber_loss_vec(  
truth,  
estimate,  
\text{delta} = 1,  
\text{na_rm} = \text{TRUE},  
\text{case_weights} = \text{NULL},  
\ldots  
)
\end{verbatim}

Arguments

- **data**: A \texttt{data.frame} containing the columns specified by the \texttt{truth} and \texttt{estimate} arguments.
- **\ldots**: Not currently used.
- **truth**: The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports \texttt{quasiquotation} (you can unquote column names). For \_vec\texttt{()} functions, a numeric vector.
- **estimate**: The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For \_vec\texttt{()} functions, a numeric vector.
huber_loss 39

delta A single numeric value. Defines the boundary where the loss function transitions from quadratic to linear. Defaults to 1.

na.rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For .vec() functions, a numeric vector.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.
For grouped data frames, the number of rows returned will be the same as the number of groups.
For huber_loss_vec(), a single numeric value (or NA).

Author(s)

James Blair

References


See Also

Other numeric metrics: ccc(), huber_loss_pseudo(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), rpd(), rpiq(), rsq_trad(), rsq(), smape()

Other accuracy metrics: ccc(), huber_loss_pseudo(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), smape()

Examples

# Supply truth and predictions as bare column names
huber_loss(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)

size <- 100

times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)
# Compute the metric by group
metric_results <- solubility_resampled %>%
group_by(resample) %>%
  huber_loss(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))

---

### huber_loss_pseudo  Psuedo-Huber Loss

**Description**

Calculate the Pseudo-Huber Loss, a smooth approximation of `huber_loss()`. Like `huber_loss()`, this is less sensitive to outliers than `rmse()`.

**Usage**

```r
huber_loss_pseudo(data, ...)
```

```r
## S3 method for class 'data.frame'
huber_loss_pseudo(
  data,
  truth,
  estimate,
  delta = 1,
  na_rm = TRUE,
  case_weights = NULL,
  ...
)
```

```r
huber_loss_pseudo_vec(
  truth,
  estimate,
  delta = 1,
  na_rm = TRUE,
  case_weights = NULL,
  ...
)
```

**Arguments**

- `data`  A `data.frame` containing the columns specified by the `truth` and `estimate` arguments.
- `...`  Not currently used.
The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.

The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.

A single numeric value. Defines the boundary where the loss function transitions from quadratic to linear. Defaults to 1.

A logical value indicating whether NA values should be stripped before the computation proceeds.

The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For huber_loss_pseudo_vec(), a single numeric value (or NA).

James Blair


Other numeric metrics: ccc(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), rpd(), rpiq(), rsq_trad(), rsq(), smape()

Other accuracy metrics: ccc(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), smape()

# Supply truth and predictions as bare column names
huber_loss_pseudo(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10
# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
  huber_loss_pseudo(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))

---

### iic

**Index of ideality of correlation**

**Description**

Calculate the index of ideality of correlation. This metric has been studied in QSPR/QSAR models as a good criterion for the predictive potential of these models. It is highly dependent on the correlation coefficient as well as the mean absolute error.

Note the application of IIC is useless under two conditions:

- When the negative mean absolute error and positive mean absolute error are both zero.
- When the outliers are symmetric. Since outliers are context dependent, please use your own checks to validate whether this restriction holds and whether the resulting IIC has interpretative value.

The IIC is seen as an alternative to the traditional correlation coefficient and is in the same units as the original data.

**Usage**

```r
iic(data, ...)
``` 

## S3 method for class 'data.frame'
```r
iic(data, truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
``` 

```r
iic_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
```
Arguments

data  A data.frame containing the columns specified by the truth and estimate arguments.
...
truth  The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate  The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
na_rm  A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights  The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.
For grouped data frames, the number of rows returned will be the same as the number of groups.
For iic_vec(), a single numeric value (or NA).

Author(s)

Joyce Cahoon

References


See Also

Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), rpd(), rpiq(), rsq_trad(), rsq(), smape()

Other accuracy metrics: ccc(), huber_loss_pseudo(), huber_loss(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), smape()

Examples

# Supply truth and predictions as bare column names
iic(solubility_test, solubility, prediction)
library(dplyr)
```r
set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
    replicate(
        n = times,
        expr = sample_n(solubility_test, size, replace = TRUE),
        simplify = FALSE
    ),
    .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
    group_by(resample) %>%
    iic(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
    summarise(avg_estimate = mean(.estimate))
```

---

### j_index

<table>
<thead>
<tr>
<th><strong>j_index</strong></th>
<th><strong>J-index</strong></th>
</tr>
</thead>
</table>

**Description**

Youden’s J statistic is defined as:

\[
\text{sens()} + \text{spec()} - 1
\]

A related metric is Informedness, see the Details section for the relationship.

**Usage**

```r
j_index(data, ...)```

## S3 method for class 'data.frame'

```r
j_index(
    data,
    truth,
    estimate,
    estimator = NULL,
    na_rm = TRUE,
    case_weights = NULL,
    event_level = yardstick_event_level(),
    ...)
```
j_index

)

j_index_vec(
  truth,
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

Arguments

data Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.

... Not currently used.

truth The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

estimate The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

estimator One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

event_level A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

Details

The value of the J-index ranges from [0, 1] and is 1 when there are no false positives and no false negatives.

The binary version of J-index is equivalent to the binary concept of Informedness. Macro-weighted J-index is equivalent to multiclass informedness as defined in Powers, David M W (2011), equation (42).
Value

A tibble with columns `.metric`, `.estimator`, and `.estimate` and 1 row of values.
For grouped data frames, the number of rows returned will be the same as the number of groups.
For `j_index_vec()`, a single numeric value (or `NA`).

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument `event_level` to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

Macro, micro, and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.

Author(s)

Max Kuhn

References


See Also

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `kap()`, `mcc()`, `npv()`, `ppv()`, `precision()`, `recall()`, `sens()`, `spec()`

Examples

# Two class
data("two_class_example")
j_index(two_class_example, truth, predicted)

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
j_index(obs, pred)
Kappa is a similar measure to `accuracy()`, but is normalized by the accuracy that would be expected by chance alone and is very useful when one or more classes have large frequency distributions.

**Usage**

```r
kap(data, ...)```

```r
# S3 method for class 'data.frame'
kap(
  data,
  truth,
  estimate,
  weighting = "none",
  na_rm = TRUE,
  case_weights = NULL,
  ...
)
```

```r
kap_vec(
  truth,
  ...)```
estimate,
weighting = "none",
na_rm = TRUE,
case_weights = NULL,
...)

Arguments

data Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.

... Not currently used.

truth The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

estimate The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

weighting A weighting to apply when computing the scores. One of: "none", "linear", or "quadratic". Linear and quadratic weighting penalizes mis-predictions that are "far away" from the true value. Note that distance is judged based on the ordering of the levels in truth and estimate. It is recommended to provide ordered factors for truth and estimate to explicitly code the ordering, but this is not required.

In the binary case, all 3 weightings produce the same value, since it is only ever possible to be 1 unit away from the true value.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For kap_vec(), a single numeric value (or NA).

Multiclass

Kappa extends naturally to multiclass scenarios. Because of this, macro and micro averaging are not implemented.
Author(s)

Max Kuhn
Jon Harmon

References


See Also

Other class metrics: accuracy(), bal_accuracy(), detection_prevalence(), f_meas(), j_index(), mcc(), npv(), ppv(), precision(), recall(), sens(), spec()

Examples

library(dplyr)
data("two_class_example")
data("hpc_cv")

# Two class
kap(two_class_example, truth, predicted)

# Multiclass
# kap() has a natural multiclass extension
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  kap(obs, pred)

# Groups are respected
hpc_cv %>%
group_by(Resample) %>%
kap(obs, pred)

lift_curve

Description

lift_curve() constructs the full lift curve and returns a tibble. See gain_curve() for a closely related concept.
Usage

lift_curve(data, ...)

# S3 method for class 'data.frame'
lift_curve(
  data,
  truth,
  ..., 
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL
)

Arguments

data

A data.frame containing the columns specified by truth and ....

... A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.

truth

The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

na_rm

A logical value indicating whether NA values should be stripped before the computation proceeds.

event_level A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

case_weights

The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Details

There is a ggplot2::autoplot() method for quickly visualizing the curve. This works for binary and multiclass output, and also works with grouped data (i.e. from resamples). See the examples.

Value

A tibble with class lift_df or lift_grouped_df having columns:

- .n The index of the current sample.
lift_curve

• `.n_events` The index of the current unique sample. Values with repeated `estimate` values are given identical indices in this column.
• `.percent_tested` The cumulative percentage of values tested.
• `.lift` First calculate the cumulative percentage of true results relative to the total number of true results. Then divide that by `.percent_tested`.

If using the `case_weights` argument, all of the above columns will be weighted. This makes the most sense with frequency weights, which are integer weights representing the number of times a particular observation should be repeated.

**Gain and Lift Curves**

The motivation behind cumulative gain and lift charts is as a visual method to determine the effectiveness of a model when compared to the results one might expect without a model. As an example, without a model, if you were to advertise to a random 10% of your customer base, then you might expect to capture 10% of the total number of positive responses had you advertised to your entire customer base. Given a model that predicts which customers are more likely to respond, the hope is that you can more accurately target 10% of your customer base and capture >10% of the total number of positive responses.

The calculation to construct lift curves is as follows:

1. truth and estimate are placed in descending order by the `estimate` values (`estimate` here is a single column supplied in ...).
2. The cumulative number of samples with true results relative to the entire number of true results are found.
3. The cumulative % found is divided by the cumulative % tested to construct the lift value. This ratio represents the factor of improvement over an uninformed model. Values >1 represent a valuable model. This is the y-axis of the lift chart.

**Multiclass**

If a multiclass `truth` column is provided, a one-vs-all approach will be taken to calculate multiple curves, one per level. In this case, there will be an additional column, `.level`, identifying the "one" column in the one-vs-all calculation.

**Relevant Level**

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In `yardstick`, the default is to use the first level. To alter this, change the argument `event_level` to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

**Author(s)**

Max Kuhn
See Also

Other curve metrics: `gain_curve()`, `pr_curve()`, `roc_curve()`

Examples

```r
# Two class example

# 'truth' is a 2 level factor. The first level is "Class1", which is the
# 'event of interest' by default in yardstick. See the Relevant Level
# section above.
data(two_class_example)

# Binary metrics using class probabilities take a factor 'truth' column,
# and a single class probability column containing the probabilities of
# the event of interest. Here, since "Class1" is the first level of
# 'truth', it is the event of interest and we pass in probabilities for it.
lift_curve(two_class_example, truth, Class1)

# 'autoplot()'

library(ggplot2)
library(dplyr)

# Use autoplot to visualize
autoplot(lift_curve(two_class_example, truth, Class1))

# Multiclass one-vs-all approach
# One curve per level
hpc_cv %>%
  filter(Resample == "Fold01") %>%
lift_curve(obs, VF:L) %>%
  autoplot()

# Same as above, but will all of the resamples
hpc_cv %>%
  group_by(Resample) %>%
lift_curve(obs, VF:L) %>%
  autoplot()
```

Survival Analysis Results

Description

Survival Analysis Results
Details

These data contain plausible results from applying predictive survival models to the lung data set using the censored package.

Value

lung_surv a data frame

Examples

data(lung_surv)
str(lung_surv)

# `surv_obj` is a `Surv()` object

mae Mean absolute error

Description

Calculate the mean absolute error. This metric is in the same units as the original data.

Usage

mae(data, ...)

## S3 method for class 'data.frame'
mae(data, truth, estimate, na_rm = TRUE, case_weights = NULL, ...)

mae_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)

Arguments

data A data.frame containing the columns specified by the truth and estimate arguments.
...
Not currently used.
truth The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.
Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For mae_vec(), a single numeric value (or NA).

Author(s)

Max Kuhn

See Also

Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), rpd(), rpiq(), rsq_trad(), rsq(), smape()

Other accuracy metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), rpiq(), rsq_trad(), rsq(), smape()

Examples

# Supply truth and predictions as bare column names
mae(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
  mae(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))
**Description**

Calculate the mean absolute percentage error. This metric is in *relative units*.

**Usage**

```r
mape(data, ...)  # S3 method for class 'data.frame'
mape(data, truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
mape_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
```

**Arguments**

- `data`: A data.frame containing the columns specified by the `truth` and `estimate` arguments.
- `...`: Not currently used.
- `truth`: The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a numeric vector.
- `estimate`: The column identifier for the predicted results (that is also numeric). As with `truth` this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a numeric vector.
- `na_rm`: A logical value indicating whether NA values should be stripped before the computation proceeds.
- `case_weights`: The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector.

**Details**

Note that a value of Inf is returned for `mape()` when the observed value is negative.

**Value**

A tibble with columns `.metric`, `.estimator`, and `.estimate` and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For `mape_vec()`, a single numeric value (or NA).
Author(s)
Max Kuhn

See Also
Other numeric metrics: `ccc()`, `huber_loss_pseudo()`, `huber_loss()`, `iic()`, `mae()`, `mase()`, `mpe()`, `msd()`, `poisson_log_loss()`, `rmse()`, `rpd()`, `rpiq()`, `rsq_trad()`, `rsq()`, `smape()`.

Other accuracy metrics: `ccc()`, `huber_loss_pseudo()`, `huber_loss()`, `iic()`, `mae()`, `mase()`, `mpe()`, `msd()`, `poisson_log_loss()`, `rmse()`, `smape()`.

Examples

```r
# Supply truth and predictions as bare column names
mape(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
  mape(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))
```

---

**mase**

*Mean absolute scaled error*

**Description**

Calculate the mean absolute scaled error. This metric is *scale independent* and *symmetric*. It is generally used for comparing forecast error in time series settings. Due to the time series nature of this metric, it is necessary to order observations in ascending order by time.
Usage

mase(data, ...)

## S3 method for class 'data.frame'
mase(
  data,
  truth,
  estimate,
  m = 1L,
  mae_train = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  ...
)

mase_vec(
  truth,
  estimate,
  m = 1L,
  mae_train = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  ...
)

Arguments

data A data.frame containing the columns specified by the truth and estimate arguments.
...
Not currently used.
truth The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
m An integer value of the number of lags used to calculate the in-sample seasonal naive error. The default is used for non-seasonal time series. If each observation was at the daily level and the data showed weekly seasonality, then m = 7L would be a reasonable choice for a 7-day seasonal naive calculation.
mae_train A numeric value which allows the user to provide the in-sample seasonal naive mean absolute error. If this value is not provided, then the out-of-sample seasonal naive mean absolute error will be calculated from truth and will be used instead.
na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights  The optional column identifier for case weights. This should be an unquoted
column name that evaluates to a numeric column in data. For _vec() functions,
a numeric vector.

Details
mase() is different from most numeric metrics. The original implementation of mase() calls for
using the in-sample naive mean absolute error to compute scaled errors with. It uses this instead
of the out-of-sample error because there is a chance that the out-of-sample error cannot be com-
puting when forecasting a very short horizon (i.e. the out of sample size is only 1 or 2). However,
yardstick only knows about the out-of-sample truth and estimate values. Because of this, the
out-of-sample error is used in the computation by default. If the in-sample naive mean absolute er-
ror is required and known, it can be passed through in the mae_train argument and it will be used
instead. If the in-sample data is available, the naive mean absolute error can easily be computed
with mae(data, truth, lagged_truth).

Value
A tibble with columns .metric, .estimator, and .estimate and 1 row of values.
For grouped data frames, the number of rows returned will be the same as the number of groups.
For mase_vec(), a single numeric value (or NA).

Author(s)
Alex Hallam

References
Rob J. Hyndman (2006). ANOTHER LOOK AT FORECAST-ACCURACY METRICS FOR IN-
TERMITTENT DEMAND. Foresight, 4, 46.

See Also
Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(),
mpe(), msd(), poisson_log_loss(), rmse(), rpd(), rpiq(), rsq_trad(), rsq(), smape()
Other accuracy metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(),
mpe(), msd(), poisson_log_loss(), rmse(), smape()

Examples
# Supply truth and predictions as bare column names
mase(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
  mase(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))

mcc

Matthews correlation coefficient

Description
Matthews correlation coefficient

Usage
mcc(data, ...)

## S3 method for class 'data.frame'
mcc(data, truth, estimate, na.rm = TRUE, case_weights = NULL, ...)

mcc_vec(truth, estimate, na.rm = TRUE, case_weights = NULL, ...)

Arguments
data
Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.

... Not currently used.

truth
The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

estimate
The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
na_rm: A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights: The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For mcc_vec(), a single numeric value (or NA).

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

mcc() has a known multiclass generalization and that is computed automatically if a factor with more than 2 levels is provided. Because of this, no averaging methods are provided.

Author(s)

Max Kuhn

References


See Also

Other class metrics: accuracy(), bal_accuracy(), detection_prevalence(), f_meas(), j_index(), kap(), npv(), ppv(), precision(), recall(), sens(), spec()

Examples

library(dplyr)
data("two_class_example")
data("hpc_cv")

# Two class
mcc(two_class_example, truth, predicted)

# Multiclass
# mcc() has a natural multiclass extension
hpc_cv %>%
  filter(Resample == "Fold01") %>%
mcc(obs, pred)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
mcc(obs, pred)

---

**metric-summarizers**  
*Developer function for summarizing new metrics*

**Description**

`numeric_metric_summarizer()`, `class_metric_summarizer()`, `prob_metric_summarizer()`, `curve_metric_summarizer()`, `dynamic_survival_metric_summarizer()`, and `static_survival_metric_summarizer()` are useful alongside `check_metric` and `yardstick_remove_missing` for implementing new custom metrics. These functions call the metric function inside `dplyr::summarise()` or `dplyr::reframe()` for `curve_metric_summarizer()`. See [Custom performance metrics](#) for more information.

**Usage**

```r
numeric_metric_summarizer(
  name,
  fn,
  data,
  truth,
  estimate,
  ...
)
```

```r
class_metric_summarizer(
  name,
  fn,
  data,
  truth,
  estimate,
  ...
)
```

---
error_call = caller_env()
)

prob_metric_summarizer(
    name,
    fn,
    data,
    truth,
    ..., 
    estimator = NULL,
    na_rm = TRUE,
    event_level = NULL,
    case_weights = NULL,
    fn_options = list(),
    error_call = caller_env()
)

curve_metric_summarizer(
    name,
    fn,
    data,
    truth,
    ..., 
    estimator = NULL,
    na_rm = TRUE,
    event_level = NULL,
    case_weights = NULL,
    fn_options = list(),
    error_call = caller_env()
)

dynamic_survival_metric_summarizer(
    name,
    fn,
    data,
    truth,
    ..., 
    na_rm = TRUE,
    case_weights = NULL,
    fn_options = list(),
    error_call = caller_env()
)

static_survival_metric_summarizer(
    name,
    fn,
    data,
    truth,
metric-summarizers

estimate,
..., 
na_rm = TRUE,
case_weights = NULL, 
fn_options = list(), 
error_call = caller_env()
)

curve_survival_metric_summarizer(
  name, 
  fn,
  data,
  truth,
  ..., 
na_rm = TRUE,
case_weights = NULL,
fn_options = list(), 
error_call = caller_env()
) 

Arguments

name A single character representing the name of the metric to use in the tibble output. This will be modified to include the type of averaging if appropriate.

fn The vector version of your custom metric function. It generally takes truth, estimate, na_rm, and any other extra arguments needed to calculate the metric.

data The data frame with truth and estimate columns passed in from the data frame version of your metric function that called numeric_metric_summarizer(), class_metric_summarizer(), prob_metric_summarizer(), curve_metric_summarizer(), dynamic_survival_metric_summarizer(), or static_survival_metric_summarizer().

truth The unquoted column name corresponding to the truth column.

estimate Generally, the unquoted column name corresponding to the estimate column. For metrics that take multiple columns through ... like class probability metrics, this is a result of dots_to_estimate().

... These dots are for future extensions and must be empty.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds. The removal is executed in yardstick_remove_missing().

case_weights For metrics supporting case weights, an unquoted column name corresponding to case weights can be passed here. If not NULL, the case weights will be passed on to fn as the named argument case_weights.

fn_options A named list of metric specific options. These are spliced into the metric function call using !!! from rlang. The default results in nothing being spliced into the call.

error_call The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
estimator

This can either be NULL for the default auto-selection of averaging (“binary” or “macro”), or a single character to pass along to the metric implementation describing the kind of averaging to use.

event_level

This can either be NULL to use the default event_level value of the fn or a single string of either "first" or "second" to pass along describing which level should be considered the "event".

Details

numeric_metric_summarizer(), class_metric_summarizer(), prob_metric_summarizer(), curve_metric_summarizer(), dynamic_survival_metric_summarizer(), and dynamic_survival_metric_summarizer() are generally called from the data frame version of your metric function. It knows how to call your metric over grouped data frames and returns a tibble consistent with other metrics.

See Also

calculate_metric yardstick_remove_missing finalize_estimator() dots_to_estimate()

metrics

General Function to Estimate Performance

Description

This function estimates one or more common performance estimates depending on the class of truth (see Value below) and returns them in a three column tibble.

Usage

metrics(data, ...)

## S3 method for class 'data.frame'
metrics(data, truth, estimate, ..., na_rm = TRUE, options = list())

Arguments

data A data.frame containing the columns specified by truth, estimate, and ....

... A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.

truth The column identifier for the true results (that is numeric or factor). This should be an unquoted column name although this argument is passed by expression and support quasiquotation (you can unquote column names).

estimate The column identifier for the predicted results (that is also numeric or factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name.
metrics

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

options [deprecated]
No longer supported as of yardstick 1.0.0. If you pass something here it will be ignored with a warning.
Previously, these were options passed on to pROC::roc(). If you need support for this, use the pROC package directly.

Value

A three column tibble.

- When truth is a factor, there are rows for accuracy() and the Kappa statistic (kap()).
- When truth has two levels and 1 column of class probabilities is passed to . . . , there are rows for the two class versions of mn_log_loss() and roc_auc().
- When truth has more than two levels and a full set of class probabilities are passed to . . . , there are rows for the multiclass version of mn_log_loss() and the Hand Till generalization of roc_auc().
- When truth is numeric, there are rows for rmse(), rsq(), and mae().

See Also

metric_set()

Examples

# Accuracy and kappa
metrics(two_class_example, truth, predicted)

# Add on multinomal log loss and ROC AUC by specifying class prob columns
metrics(two_class_example, truth, predicted, Class1)

# Regression metrics
metrics(solubility_test, truth = solubility, estimate = prediction)

# Multiclass metrics work, but you cannot specify any averaging
# for roc_auc() besides the default, hand_till. Use the specific function
# if you need more customization
library(dplyr)

hpc_cv %>%
group_by(Resample) %>%
metrics(obs, pred, VF:L) %>%
print(n = 40)
metric_set

Combine metric functions

Description

metric_set() allows you to combine multiple metric functions together into a new function that calculates all of them at once.

Usage

metric_set(...)  

Arguments

...  

The bare names of the functions to be included in the metric set.

Details

All functions must be either:

- Only numeric metrics
- A mix of class metrics or class prob metrics
- A mix of dynamic, integrated, and static survival metrics

For instance, rmse() can be used with mae() because they are numeric metrics, but not with accuracy() because it is a classification metric. But accuracy() can be used with roc_auc().

The returned metric function will have a different argument list depending on whether numeric metrics or a mix of class/prob metrics were passed in.

# Numeric metric set signature:
fn(
    data,
    truth,
    estimate,
    na_rm = TRUE,
    case_weights = NULL,
    ...
)

# Class / prob metric set signature:
fn(
    data,
    truth,
    ..., 
    estimate,
    estimator = NULL,
    na_rm = TRUE,
# Dynamic / integrated / static survival metric set signature:
fn(
  data,
  truth,
  ..., 
  estimate,
  na.rm = TRUE,
  case_weights = NULL 
)

When mixing class and class prob metrics, pass in the hard predictions (the factor column) as the named argument `estimate`, and the soft predictions (the class probability columns) as bare column names or `tidyselect` selectors to ....

When mixing dynamic, integrated, and static survival metrics, pass in the time predictions as the named argument `estimate`, and the survival predictions as bare column names or `tidyselect` selectors to ....

If `metric_tweak()` has been used to "tweak" one of these arguments, like `estimator` or `event_level`, then the tweaked version wins. This allows you to set the estimator on a metric by metric basis and still use it in a `metric_set()`.

### See Also

- `metrics()`

### Examples

```r
library(dplyr)

# Multiple regression metrics
multi_metric <- metric_set(rmse, rsq, ccc)

# The returned function has arguments:
# fn(data, truth, estimate, na.rm = TRUE, ...)
multi_metric(solubility_test, truth = solubility, estimate = prediction)

# Groups are respected on the new metric function
class_metrics <- metric_set(accuracy, kap)

hpc_cv %>%
  group_by(Resample) %>%
  class_metrics(obs, estimate = pred)

# If you need to set options for certain metrics,
# do so by wrapping the metric and setting the options inside the wrapper,
```
# passing along truth and estimate as quoted arguments.
# Then add on the function class of the underlying wrapped function,
# and the direction of optimization.
ccc_with_bias <- function(data, truth, estimate, na_rm = TRUE, ...) {
  ccc(
    data = data,
    truth = !!!rlang::enquo(truth),
    estimate = !!!rlang::enquo(estimate),
    # set bias = TRUE
    bias = TRUE,
    na_rm = na_rm,
    ...
  )
}

# Use `new_numeric_metric()` to formalize this new metric function
ccc_with_bias <- new_numeric_metric(ccc_with_bias, "maximize")

multi_metric2 <- metric_set(rmse, rsq, ccc_with_bias)

multi_metric2(solubility_test, truth = solubility, estimate = prediction)

# A class probability example:
# Note that, when given class or class prob functions,
# metric_set() returns a function with signature:
# fn(data, truth, ..., estimate)
# to be able to mix class and class prob metrics.
#
# You must provide the `estimate` column by explicitly naming
# the argument

class_and_probs_metrics <- metric_set(roc_auc, pr_auc, accuracy)

hpc_cv %>%
  group_by(Resample) %>%
class_and_probs_metrics(obs, VF:L, estimate = pred)

---

**Description**

`metric_tweak()` allows you to tweak an existing metric `.fn`, giving it a new `.name` and setting new optional argument defaults through `...`. It is similar to `purrr::partial()`, but is designed specifically for yardstick metrics.

`metric_tweak()` is especially useful when constructing a `metric_set()` for tuning with the tune package. After the metric set has been constructed, there is no way to adjust the value of any
optional arguments (such as beta in \texttt{f_meas}). Using \texttt{metric_tweak()}, you can set optional arguments to custom values ahead of time, before they go into the metric set.

\textbf{Usage}

\texttt{metric_tweak(.name, .fn, ...)}

\textbf{Arguments}

\begin{itemize}
\item \texttt{.name} \quad \texttt{A single string giving the name of the new metric. This will be used in the ".metric" column of the output.}
\item \texttt{.fn} \quad \texttt{An existing yardstick metric function to tweak.}
\item \texttt{...} \quad \texttt{Name-value pairs specifying which optional arguments to override and the values to replace them with.}
\end{itemize}

Arguments \texttt{data}, \texttt{truth}, and \texttt{estimate} are considered \texttt{protected}, and cannot be overridden, but all other optional arguments can be altered.

\textbf{Details}

The function returned from \texttt{metric_tweak()} only takes \texttt{...} as arguments, which are passed through to the original \texttt{.fn}. Passing \texttt{data}, \texttt{truth}, and \texttt{estimate} through by position should generally be safe, but it is recommended to pass any other optional arguments through by name to ensure that they are evaluated correctly.

\textbf{Value}

A tweaked version of \texttt{.fn}, updated to use new defaults supplied in \texttt{...}.

\textbf{Examples}

\begin{verbatim}
mase12 <- metric_tweak("mase12", mase, m = 12)

# Defaults to 'm = 1'
mase(solubility_test, solubility, prediction)

# Updated to use 'm = 12'. 'mase12()' has this set already.
mase(solubility_test, solubility, prediction, m = 12)
mase12(solubility_test, solubility, prediction)

# This is most useful to set optional argument values ahead of time when
# using a metric set
mase10 <- metric_tweak("mase10", mase, m = 10)
metrics <- metric_set(mase, mase10, mase12)
metrics(solubility_test, solubility, prediction)
\end{verbatim}
**mn_log_loss**  
*Mean log loss for multinomial data*

**Description**

Compute the logarithmic loss of a classification model.

**Usage**

```r
mn_log_loss(data, ...) 
```

```r
# S3 method for class 'data.frame'
mn_log_loss(
  data,
  truth,
  ..., 
  na.rm = TRUE,
  sum = FALSE,
  event_level = yardstick_event_level(),
  case_weights = NULL
)
```

```r
mn_log_loss_vec(
  truth,
  estimate,
  na_rm = TRUE,
  sum = FALSE,
  event_level = yardstick_event_level(),
  case_weights = NULL,
  ...
)
```

**Arguments**

- **data** A `data.frame` containing the columns specified by `truth` and `...`
- **...** A set of unquoted column names or one or more `dplyr` selector functions to choose which variables contain the class probabilities. If `truth` is binary, only 1 column should be selected, and it should correspond to the value of `event_level`. Otherwise, there should be as many columns as factor levels of `truth` and the ordering of the columns should be the same as the factor levels of `truth`.
- **truth** The column identifier for the true class results (that is a `factor`). This should be an unquoted column name although this argument is passed by expression and supports `quasiquotation` (you can unquote column names). For `__vec()` functions, a factor vector.
- **na_rm** A logical value indicating whether NA values should be stripped before the computation proceeds.
mn_log_loss

sum
A logical. Should the sum of the likelihood contributions be returned (instead of the mean value)?

event_level
A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

case_weights
The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

estimate
If truth is binary, a numeric vector of class probabilities corresponding to the "relevant" class. Otherwise, a matrix with as many columns as factor levels of truth. It is assumed that these are in the same order as the levels of truth.

Details
Log loss is a measure of the performance of a classification model. A perfect model has a log loss of 0.

Compared with accuracy(), log loss takes into account the uncertainty in the prediction and gives a more detailed view into the actual performance. For example, given two input probabilities of .6 and .9 where both are classified as predicting a positive value, say, "Yes", the accuracy metric would interpret them as having the same value. If the true output is "Yes", log loss penalizes .6 because it is "less sure" of it's result compared to the probability of .9.

Value
A tibble with columns .metric, .estimator, and .estimate and 1 row of values.
For grouped data frames, the number of rows returned will be the same as the number of groups.
For mn_log_loss_vec(), a single numeric value (or NA).

Multiclass
Log loss has a known multiclass extension, and is simply the sum of the log loss values for each class prediction. Because of this, no averaging types are supported.

Author(s)
Max Kuhn

See Also
Other class probability metrics: average_precision(), brier_class(), classification_cost(), gain_capture(), pr_auc(), roc_auc(), roc_aunp(), roc_aunu()
Examples

# Two class
data("two_class_example")
mn_log_loss(two_class_example, truth, Class1)

# Multiclass
library(dplyr)
data(hpc_cv)

# You can use the col1:colN tidyselect syntax
hpc_cv %>%
  filter(Resample == "Fold01") %>%
mn_log_loss(obs, VF:L)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
mn_log_loss(obs, VF:L)

# Vector version
# Supply a matrix of class probabilities
fold1 <- hpc_cv %>%
  filter(Resample == "Fold01")

mn_log_loss_vec(
  truth = fold1$obs,
  matrix(,
    c(fold1$VF, fold1$F, fold1$M, fold1$L),
    ncol = 4
  )
)

# Supply `...` with quasiquotation
prob_cols <- levels(two_class_example$truth)
mn_log_loss(two_class_example, truth, Class1)
mn_log_loss(two_class_example, truth, !!prob_cols[[1]])

mpe

Mean percentage error

Description

Calculate the mean percentage error. This metric is in relative units. It can be used as a measure of the estimate's bias.

Note that if any truth values are 0, a value of: -Inf (estimate > 0), Inf (estimate < 0), or NaN (estimate == 0) is returned for mpe().
mpe

Usage

mpe(data, ...)

## S3 method for class 'data.frame'
mpe(data, truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
mpe_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)

Arguments

data A data.frame containing the columns specified by the truth and estimate arguments.

... Not currently used.

truth The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.

estimate The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For mpe_vec(), a single numeric value (or NA).

Author(s)

Thomas Bierhance

See Also

Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), msd(), poisson_log_loss(), rmse(), rpd(), rpiq(), rsq_trad(), rsq(), smape()

Other accuracy metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), msd(), poisson_log_loss(), rmse(), smape()
Examples

```r
# 'solubility_test$solubility' has zero values with corresponding
# 'prediction' values that are negative. By definition, this causes 'Inf'
# to be returned from 'mpe()'.
solubility_test[solubility_test$solubility == 0, ]
mpe(solubility_test, solubility, prediction)

# We'll remove the zero values for demonstration
solubility_test <- solubility_test[solubility_test$solubility != 0, ]

# Supply truth and predictions as bare column names
mpe(solubility_test, solubility, prediction)

library(dplyr)
set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
mpe(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))
```

---

**msd**

*Mean signed deviation*

**Description**

Mean signed deviation (also known as mean signed difference, or mean signed error) computes
the average differences between truth and estimate. A related metric is the mean absolute error
(*mae()*).
msd

Usage

msd(data, ...)

## S3 method for class 'data.frame'
msd(data, truth, estimate, na.rm = TRUE, case_weights = NULL, ...)

msd_vec(truth, estimate, na.rm = TRUE, case_weights = NULL, ...)

Arguments

data A data.frame containing the columns specified by the truth and estimate arguments.

... Not currently used.

truth The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.

estimate The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Details

Mean signed deviation is rarely used, since positive and negative errors cancel each other out. For example, msd_vec(c(100, -100), c(0, 0)) would return a seemingly "perfect" value of 0, even though estimate is wildly different from truth. mae() attempts to remedy this by taking the absolute value of the differences before computing the mean.

This metric is computed as mean(truth - estimate), following the convention that an "error" is computed as observed - predicted. If you expected this metric to be computed as mean(estimate - truth), reverse the sign of the result.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For msd_vec(), a single numeric value (or NA).

Author(s)

Thomas Bierhance
new-metric

See Also

Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), poisson_log_loss(), rmse(), rpd(), rpiq(), rsq_trad(), rsq(), smape()

Other accuracy metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), poisson_log_loss(), rmse(), smape()

Examples

# Supply truth and predictions as bare column names
msd(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
   replicate(
      n = times,
      expr = sample_n(solubility_test, size, replace = TRUE),
      simplify = FALSE
   ),
   .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
   group_by(resample) %>%
   msd(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
   summarise(avg_estimate = mean(.estimate))

new-metric

Construct a new metric function

Description

These functions provide convenient wrappers to create the three types of metric functions in yardstick: numeric metrics, class metrics, and class probability metrics. They add a metric-specific class to fn and attach a direction attribute. These features are used by metric_set() and by tune when model tuning.

See Custom performance metrics for more information about creating custom metrics.
Usage

new_class_metric(fn, direction)
new_prob_metric(fn, direction)
new_numeric_metric(fn, direction)
new_dynamic_survival_metric(fn, direction)
new_integrated_survival_metric(fn, direction)
new_static_survival_metric(fn, direction)

Arguments

fn A function. The metric function to attach a metric-specific class and direction attribute to.
direction A string. One of:
  • "maximize"
  • "minimize"
  • "zero"

Description

These functions calculate the npv() (negative predictive value) of a measurement system compared to a reference result (the "truth" or gold standard). Highly related functions are spec(), sens(), andppv().

Usage

npv(data, ...)

## S3 method for class 'data.frame'
npv(
  data,
  truth,
  estimate,
  prevalence = NULL,
  estimator = NULL,
  na.rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)
npv_vec(
    truth,
    estimate,
    prevalence = NULL,
    estimator = NULL,
    na_rm = TRUE,
    case_weights = NULL,
    event_level = yardstick_event_level(),
    ...
)

Arguments

data Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.

... Not currently used.

truth The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

estimate The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

prevalence A numeric value for the rate of the "positive" class of the data.

estimator One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

event_level A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

Details

The positive predictive value (ppv()) is defined as the percent of predicted positives that are actually positive while the negative predictive value (npv()) is defined as the percent of negative positives that are actually negative.
**Value**

A tibble with columns `.metric`, `.estimator`, and `.estimate` and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For npv_vec(), a single numeric value (or NA).

**Relevant Level**

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument `event_level` to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

**Multiclass**

Macro, micro, and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.

**Implementation**

Suppose a 2x2 table with notation:

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>A</td>
</tr>
<tr>
<td>Negative</td>
<td>C</td>
</tr>
<tr>
<td>Positive</td>
<td>B</td>
</tr>
<tr>
<td>Negative</td>
<td>D</td>
</tr>
</tbody>
</table>

The formulas used here are:

\[
Sensitivity = \frac{A}{A + C}
\]

\[
Specificity = \frac{D}{B + D}
\]

\[
Prevalence = \frac{A + C}{A + B + C + D}
\]

\[
PPV = \frac{(Sensitivity \times Prevalence) + ((1 - Specificity) \times (1 - Prevalence))}{(Sensitivity \times Prevalence) + ((1 - Specificity) \times (1 - Prevalence))}
\]

\[
NPV = \frac{(Specificity \times (1 - Prevalence)) + ((1 - Sensitivity) \times Prevalence)}{((1 - Sensitivity) \times Prevalence) + ((Specificity) \times (1 - Prevalence))}
\]

See the references for discussions of the statistics.

**Author(s)**

Max Kuhn
References


See Also

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `j_index()`, `kap()`, `mcc()`, `ppv()`, `precision()`, `recall()`, `sens()`, `spec()`

Other sensitivity metrics: `ppv()`, `sens()`, `spec()`

Examples

# Two class
data("two_class_example")
npv(two_class_example, truth, predicted)

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
npv(obs, pred)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
npv(obs, pred)

# Weighted macro averaging
hpc_cv %>%
  group_by(Resample) %>%
npv(obs, pred, estimator = "macro_weighted")

# Vector version
npv_vec(
  two_class_example$truth,
  two_class_example$predicted
)

# Making Class2 the "relevant" level
npv_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)
Liver Pathology Data

Description

Liver Pathology Data

Details

These data have the results of an x-ray examination to determine whether liver is abnormal or not (in the scan column) versus the more extensive pathology results that approximate the truth (in pathology).

Value

- **pathology**: a data frame

Source


Examples

```r
data(pathology)
str(pathology)
```

---

Mean log loss for Poisson data

Description

Calculate the loss function for the Poisson distribution.

Usage

```r
poisson_log_loss(data, ...)
# S3 method for class 'data.frame'
poisson_log_loss(data, truth, estimate, na.rm = TRUE, case_weights = NULL, ...)
poisson_log_loss_vec(truth, estimate, na.rm = TRUE, case_weights = NULL, ...)
```
Arguments

data A data.frame containing the columns specified by the truth and estimate arguments.

... Not currently used.

truth The column identifier for the true counts (that is integer). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, an integer vector.

estimate The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For poisson_log_loss_vec(), a single numeric value (or NA).

Author(s)

Max Kuhn

See Also

Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), rmse(), rpd(), rpiq(), rsq_trad(), rsq(), smape()

Other accuracy metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), rmse(), smape()

Examples

count_truth <- c(2L, 7L, 1L, 1L, 0L, 3L)
count_pred <- c(2.14, 5.35, 1.65, 1.56, 1.3, 2.71)
count_results <- dplyr::tibble(count = count_truth, pred = count_pred)

# Supply truth and predictions as bare column names
poisson_log_loss(count_results, count, pred)
**ppv**

Positive predictive value

---

**Description**

These functions calculate the `ppv()` (positive predictive value) of a measurement system compared to a reference result (the "truth" or gold standard). Highly related functions are `spec()`, `sens()`, and `npv()`.

**Usage**

```r
ppv(data, ...)  
## S3 method for class 'data.frame'
ppv(
  data,
  truth,
  estimate,
  prevalence = NULL,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...  
)
```

```r
ppv_vec(
  truth,
  estimate,
  prevalence = NULL,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)
```

**Arguments**

- `data`: Either a `data.frame` containing the columns specified by the `truth` and `estimate` arguments, or a `table/matrix` where the true class results should be in the columns of the table.
- `truth`: The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `factor` vector.
- `estimate`: The column identifier for the estimated class results (that is a factor).
- `prevalence`: The population prevalence (default: NULL)
- `estimator`: The estimator for prevalence (default: NULL)
- `na.rm`:  Logical (default: TRUE)
- `case_weights`: The case weights (default: NULL)
- `event_level`: The event level of the classification task (default: yardstick_event_level())
- `...`: Not currently used.

---
estimate  The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

prevalence  A numeric value for the rate of the "positive" class of the data.

estimator  One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

na_rm  A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights  The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

event_level  A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

Details

The positive predictive value (ppv()) is defined as the percent of predicted positives that are actually positive while the negative predictive value (npv()) is defined as the percent of negative positives that are actually negative.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For ppv_vec(), a single numeric value (or NA).

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

Macro, micro, and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.
Implementation

Suppose a 2x2 table with notation:

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Reference Positive</th>
<th>Reference Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Negative</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

The formulas used here are:

Sensitivity  =  \frac{A}{A + C}

Specificity =  \frac{D}{B + D}

Prevalence =  \frac{(A + C)}{(A + B + C + D)}

\[ PPV = \frac{(Sensitivity \times Prevalence) + ((1 - Specificity) \times (1 - Prevalence))}{(Sensitivity \times Prevalence) + ((1 - Specificity) \times (1 - Prevalence))} \]

\[ NPV = \frac{Specificity \times (1 - Prevalence)}{((1 - Sensitivity) \times Prevalence) + (Specificity \times (1 - Prevalence))} \]

See the references for discussions of the statistics.

Author(s)

Max Kuhn

References


See Also

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `j_index()`, `kap()`, `mcc()`, `npv()`, `precision()`, `recall()`, `sens()`, `spec()`

Other sensitivity metrics: `npv()`, `sens()`, `spec()`

Examples

# Two class
data("two_class_example")
ppv(two_class_example, truth, predicted)

# Multiclass
library(dplyr)
data(hpc_cv)
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  ppv(obs, pred)
# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  ppv(obs, pred)

# Weighted macro averaging
hpc_cv %>%
  group_by(Resample) %>%
  ppv(obs, pred, estimator = "macro_weighted")

# Vector version
ppv_vec(
  two_class_example$truth,
  two_class_example$predicted
)

# Making Class2 the "relevant" level
ppv_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)

# But what if we think that Class 1 only occurs 40% of the time?
ppv(two_class_example, truth, predicted, prevalence = 0.40)

<table>
<thead>
<tr>
<th>precision</th>
<th>Precision</th>
</tr>
</thead>
</table>

**Description**

These functions calculate the precision() of a measurement system for finding relevant documents compared to reference results (the truth regarding relevance). Highly related functions are recall() and f_meas().

**Usage**

precision(data, ...)

## S3 method for class 'data.frame'
precision(
  data,
  truth,
  estimate,
  estimator = NULL,
  na.rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
Arguments

data  Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.

...  Not currently used.

truth  The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

estimate  The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

estimator  One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

na_rm  A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights  The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

event_level  A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

Details

The precision is the percentage of predicted truly relevant results of the total number of predicted relevant results and characterizes the "purity in retrieval performance" (Buckland and Gey, 1994).
When the denominator of the calculation is 0, precision is undefined. This happens when both \# true_positive = 0 and \# false_positive = 0 are true, which mean that there were no predicted events. When computing binary precision, a NA value will be returned with a warning. When computing multiclass precision, the individual NA values will be removed, and the computation will proceed, with a warning.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For `precision_vec()`, a single numeric value (or NA).

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In `yardstick`, the default is to use the first level. To alter this, change the argument `event_level` to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

Macro, micro, and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.

Implementation

Suppose a 2x2 table with notation:

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Reference</th>
<th>Irrelevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Irrelevant</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

The formulas used here are:

\[
\text{recall} = \frac{A}{(A + C)}
\]

\[
\text{precision} = \frac{A}{(A + B)}
\]

\[
F_{\text{meas}} = (1 + \beta^2) \times \text{precision} \times \text{recall}/((\beta^2 \times \text{precision}) + \text{recall})
\]

See the references for discussions of the statistics.

Author(s)

Max Kuhn
References


See Also

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `j_index()`, `kap()`, `mcc()`, `npv()`, `ppv()`, `recall()`, `sens()`, `spec()`

Other relevance metrics: `f_meas()`, `recall()`

Examples

```r
# Two class
data("two_class_example")
precision(two_class_example, truth, predicted)

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
  precision(obs, pred)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  precision(obs, pred)

# Weighted macro averaging
hpc_cv %>%
  group_by(Resample) %>%
  precision(obs, pred, estimator = "macro_weighted")

# Vector version
precision_vec(
  two_class_example$truth,
  two_class_example$predicted
)

# Making Class2 the "relevant" level
precision_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)
```
pr_auc

Area under the precision recall curve

**Description**

pr_auc() is a metric that computes the area under the precision recall curve. See pr_curve() for the full curve.

**Usage**

```r
pr_auc(data, ...)
```

```
## S3 method for class 'data.frame'
pr_auc(
  data,
  truth,
  ..., 
  estimator = NULL,
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL
)
```

```
pr_auc_vec(
  truth,
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL,
  ...
)
```

**Arguments**

- `data` A data.frame containing the columns specified by truth and ....
- `truth` A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.
- `estimator` The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
pr_auc

estimator  One of "binary", "macro", or "macro_weighted" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other two are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on truth.

na_rm  A logical value indicating whether NA values should be stripped before the computation proceeds.

event_level  A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

case_weights  The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

estimate  If truth is binary, a numeric vector of class probabilities corresponding to the "relevant" class. Otherwise, a matrix with as many columns as factor levels of truth. It is assumed that these are in the same order as the levels of truth.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For pr_auc_vec(), a single numeric value (or NA).

Multiclass

Macro and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Author(s)

Max Kuhn

See Also

pr_curve() for computing the full precision recall curve.

Other class probability metrics: average_precision(), brier_class(), classification_cost(), gain_capture(), mn_log_loss(), roc_auc(), roc_aunp(), roc_aunu()
Examples

# Two class example
# `truth` is a 2 level factor. The first level is "Class1", which is the
# "event of interest" by default in yardstick. See the Relevant Level
# section above.
data(two_class_example)

# Binary metrics using class probabilities take a factor `truth` column,
# and a single class probability column containing the probabilities of
# the event of interest. Here, since "Class1" is the first level of
# "truth", it is the event of interest and we pass in probabilities for it.
pr_auc(two_class_example, truth, Class1)

# Multiclass example
# `obs` is a 4 level factor. The first level is "VF", which is the
# "event of interest" by default in yardstick. See the Relevant Level
# section above.
data(hpc_cv)

# You can use the `col1:colN` tidyselect syntax
library(dplyr)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
  pr_auc(obs, VF:L)

# Change the first level of `obs` from "VF" to "M" to alter the
# event of interest. The class probability columns should be supplied
# in the same order as the levels.
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  mutate(obs = relevel(obs, "M")) %>%
  pr_auc(obs, M, VF:L)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  pr_auc(obs, VF:L)

# Weighted macro averaging
hpc_cv %>%
  group_by(Resample) %>%
  pr_auc(obs, VF:L, estimator = "macro_weighted")

# Vector version
# Supply a matrix of class probabilities
fold1 <- hpc_cv %>%
  filter(Resample == "Fold01")
pr_curve

pr_auc_vec(
  truth = fold1$obs,
  matrix(
    c(fold1$VF, fold1$F, fold1$M, fold1$L),
    ncol = 4
  )
)

---

pr_curve  

Precision recall curve

Description

pr_curve() constructs the full precision recall curve and returns a tibble. See pr_auc() for the area under the precision recall curve.

Usage

pr_curve(data, ...)

## S3 method for class 'data.frame'
pr_curve(
  data,
  truth,
  ...,  
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL
)

Arguments

data  

A data.frame containing the columns specified by truth and ....

...  

A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.

truth  

The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

na_rm  

A logical value indicating whether NA values should be stripped before the computation proceeds.
event_level A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Details

pr_curve() computes the precision at every unique value of the probability column (in addition to infinity).

There is a ggplot2::autoplot() method for quickly visualizing the curve. This works for binary and multiclass output, and also works with grouped data (i.e. from resamples). See the examples.

Value

A tibble with class pr_df or pr_grouped_df having columns .threshold, recall, and precision.

Multiclass

If a multiclass truth column is provided, a one-vs-all approach will be taken to calculate multiple curves, one per level. In this case, there will be an additional column, .level, identifying the "one" column in the one-vs-all calculation.

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Author(s)

Max Kuhn

See Also

Compute the area under the precision recall curve with pr_auc().

Other curve metrics: gain_curve(), lift_curve(), roc_curve()

Examples

# Two class example
Recall

Description

These functions calculate the recall() of a measurement system for finding relevant documents compared to reference results (the truth regarding relevance). Highly related functions are precision() and f_meas().
Usage

recall(data, ...)

## S3 method for class 'data.frame'
recall(
  data,
  truth,
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

recall_vec(
  truth,
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

Arguments

data Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.

... Not currently used.

truth The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

estimate The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

estimator One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.
event_level  A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

Details

The recall (aka sensitivity) is defined as the proportion of relevant results out of the number of samples which were actually relevant. When there are no relevant results, recall is not defined and a value of NA is returned.

When the denominator of the calculation is 0, recall is undefined. This happens when both # true_positive = 0 and # false_negative = 0 are true, which mean that there were no true events. When computing binary recall, a NA value will be returned with a warning. When computing multiclass recall, the individual NA values will be removed, and the computation will proceed, with a warning.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For recall_vec(), a single numeric value (or NA).

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

Macro, micro, and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.

Implementation

Suppose a 2x2 table with notation:

<table>
<thead>
<tr>
<th>Predicted</th>
<th>Reference</th>
<th>Irrelevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relevant</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Irrelevant</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

The formulas used here are:
\[ recall = \frac{A}{A+C} \]
\[ precision = \frac{A}{A+B} \]
\[ F_{meas} = \frac{(1 + \beta^2) \times precision \times recall}{(\beta^2 \times precision) + recall} \]

See the references for discussions of the statistics.

**Author(s)**
Max Kuhn

**References**

**See Also**
Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `j_index()`, `kap()`, `mcc()`, `npv()`, `ppv()`, `precision()`, `sens()`, `spec()`

Other relevance metrics: `f_meas()`, `precision()`

**Examples**
```r
# Two class
data("two_class_example")
recall(two_class_example, truth, predicted)
```
```r
# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
  recall(obs, pred)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  recall(obs, pred)

# Weighted macro averaging
hpc_cv %>%
  group_by(Resample) %>%
  recall(obs, pred, estimator = "macro_weighted")

# Vector version
recall_vec(
```
two_class_example$truth,
    two_class_example$predicted
)

# Making Class2 the "relevant" level
recall_vec(
    two_class_example$truth,
    two_class_example$predicted,
    event_level = "second"
)

rmse          | Root mean squared error
---------------|--------------------------

**Description**

Calculate the root mean squared error. `rmse()` is a metric that is in the same units as the original data.

**Usage**

```r
rmse(data, ...)
```

```r
## S3 method for class 'data.frame'
rmse(data, truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
```

```r
rmse_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
```

**Arguments**

- `data` A `data.frame` containing the columns specified by the `truth` and `estimate` arguments.
- `...` Not currently used.
- `truth` The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a numeric vector.
- `estimate` The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a numeric vector.
- `na_rm` A logical value indicating whether NA values should be stripped before the computation proceeds.
- `case_weights` The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector.
Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For rmse_vec(), a single numeric value (or NA).

Author(s)

Max Kuhn

See Also

Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rpd(), rpiq(), rsq_trad(), rsq(), smape()

Other accuracy metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), smape()

Examples

# Supply truth and predictions as bare column names
rmse(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
  rmse(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))
Description

roc_auc() is a metric that computes the area under the ROC curve. See roc_curve() for the full curve.

Usage

roc_auc(data, ...)

## S3 method for class 'data.frame'
roc_auc(
  data,
  truth,
  ..., 
  estimator = NULL,
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL,
  options = list()
)

roc_auc_vec(
  truth,
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL,
  options = list(),
  ...
)

Arguments

data A data.frame containing the columns specified by truth and ...

... A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.

truth The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimator  One of "binary", "hand_till", "macro", or "macro_weighted" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The others are general methods for calculating multiclass metrics. The default will automatically choose "binary" if truth is binary, "hand_till" if truth has >2 levels and case_weights isn't specified, or "macro" if truth has >2 levels and case_weights is specified (in which case "hand_till" isn't well-defined).

na_rm  A logical value indicating whether NA values should be stripped before the computation proceeds.

event_level  A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

case_weights  The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

options  [deprecated]
No longer supported as of yardstick 1.0.0. If you pass something here it will be ignored with a warning.
Previously, these were options passed on to pROC::roc(). If you need support for this, use the pROC package directly.

estimate  If truth is binary, a numeric vector of class probabilities corresponding to the "relevant" class. Otherwise, a matrix with as many columns as factor levels of truth. It is assumed that these are in the same order as the levels of truth.

Details

Generally, an ROC AUC value is between 0.5 and 1, with 1 being a perfect prediction model. If your value is between 0 and 0.5, then this implies that you have meaningful information in your model, but it is being applied incorrectly because doing the opposite of what the model predicts would result in an AUC >0.5.

Note that you can't combine estimator = "hand_till" with case_weights.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.
For roc_auc_vec(), a single numeric value (or NA).

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.
Multiclass

The default multiclass method for computing `roc_auc()` is to use the method from Hand, Till, (2001). Unlike macro-averaging, this method is insensitive to class distributions like the binary ROC AUC case. Additionally, while other multiclass techniques will return NA if any levels in truth occur zero times in the actual data, the Hand-Till method will simply ignore those levels in the averaging calculation, with a warning.

Macro and macro-weighted averaging are still provided, even though they are not the default. In fact, macro-weighted averaging corresponds to the same definition of multiclass AUC given by Provost and Domingos (2001).

Author(s)
Max Kuhn

References


See Also

`roc_curve()` for computing the full ROC curve.

Other class probability metrics: `average_precision()`, `brier_class()`, `classification_cost()`, `gain_capture()`, `mn_log_loss()`, `pr_auc()`, `roc_aunp()`, `roc_aunu()`

Examples

```# Two class example

# `truth` is a 2 level factor. The first level is "Class1", which is the # "event of interest" by default in yardstick. See the Relevant Level # section above.
data(two_class_example)

# Binary metrics using class probabilities take a factor `truth` column, # and a single class probability column containing the probabilities of # the event of interest. Here, since "Class1" is the first level of # "truth", it is the event of interest and we pass in probabilities for it.
roc_auc(two_class_example, truth, Class1)

# Multiclass example

# `obs` is a 4 level factor. The first level is "VF", which is the```
# "event of interest" by default in yardstick. See the Relevant Level
# section above.
data(hpc_cv)

# You can use the col1:colN tidyselect syntax
library(dplyr)
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  roc_auc(obs, VF:L)

# Change the first level of `obs` from "VF" to "M" to alter the
# event of interest. The class probability columns should be supplied
# in the same order as the levels.
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  mutate(obs = relevel(obs, "M")) %>%
  roc_auc(obs, VML)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  roc_auc(obs, VF:L)

# Weighted macro averaging
hpc_cv %>%
  group_by(Resample) %>%
  roc_auc(obs, VF:L, estimator = "macro_weighted")

# Vector version
# Supply a matrix of class probabilities
fold1 <- hpc_cv %>%
  filter(Resample == "Fold01")

roc_auc_vec(
  truth = fold1$obs,
  matrix(
    c(fold1$VF, fold1$F, fold1$M, fold1$L),
    ncol = 4
  )
)

---

roc_aunp

Area under the ROC curve of each class against the rest, using the a
priori class distribution

Description

roc_aunp() is a multiclass metric that computes the area under the ROC curve of each class
against the rest, using the a priori class distribution. This is equivalent to roc_auc(estimator
= "macro_weighted").
Usage

roc_aunp(data, ...)

## S3 method for class 'data.frame'
roc_aunp(data, truth, ..., na_rm = TRUE, case_weights = NULL, options = list())

roc_aunp_vec(
  truth,
  estimate,
  na_rm = TRUE,
  case_weights = NULL,
  options = list(),
  ...
)

Arguments

data            A data.frame containing the columns specified by truth and ....
...
A set of unquoted column names or one or more dplyr selector functions to
choose which variables contain the class probabilities. There should be as many
columns as factor levels of truth.

truth          The column identifier for the true class results (that is a factor). This should be
an unquoted column name although this argument is passed by expression and
supports quasiquotation (you can unquote column names). For _vec() func-
tions, a factor vector.

na_rm           A logical value indicating whether NA values should be stripped before the
computation proceeds.

case_weights    The optional column identifier for case weights. This should be an unquoted
column name that evaluates to a numeric column in data. For _vec() functions,
a numeric vector.

options         [deprecated]
No longer supported as of yardstick 1.0.0. If you pass something here it will be
ignored with a warning.
Previously, these were options passed on to pROC::roc(). If you need support
for this, use the pROC package directly.

estimate        A matrix with as many columns as factor levels of truth. It is assumed that
these are in the same order as the levels of truth.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For roc_aunp_vec(), a single numeric value (or NA).
Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument `event_level` to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

This multiclass method for computing the area under the ROC curve uses the a priori class distribution and is equivalent to `roc_auc(estimator = "macro_weighted")`.

Author(s)

Julia Silge

References


See Also

`roc_aunu()` for computing the area under the ROC curve of each class against the rest, using the uniform class distribution.

Other class probability metrics: `average_precision()`, `brier_class()`, `classification_cost()`, `gain_capture()`, `mn_log_loss()`, `pr_auc()`, `roc_auc()`, `roc_aunu()`

Examples

# Multiclass example

# 'obs' is a 4 level factor. The first level is "VF", which is the # "event of interest" by default in yardstick. See the Relevant Level # section above.
# data(hpc_cv)

# You can use the col1:colN tidyselect syntax
library(dplyr)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
  roc_aunp(obs, VF:L)

# Change the first level of 'obs' from "VF" to "M" to alter the # event of interest. The class probability columns should be supplied # in the same order as the levels.

hpc_cv %>%
  filter(Resample == "Fold01") %>%
  mutate(obs = relevel(obs, "M")) %>%
roc_aunu(obs, M, VF:L)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  roc_aunu(obs, VF:L)

# Vector version
# Supply a matrix of class probabilities
fold1 <- hpc_cv %>%
  filter(Resample == "Fold01")

roc_aunu_vec(
  truth = fold1$obs,
  matrix(
    c(fold1$VF, fold1$F, fold1$M, fold1$L),
    ncol = 4
  )
)

---

**Description**

roc_aunu() is a multiclass metric that computes the area under the ROC curve of each class against the rest, using the uniform class distribution. This is equivalent to roc_auc(estimator = "macro").

**Usage**

roc_aunu(data, ...)

## S3 method for class 'data.frame'
roc_aunu(data, truth, ..., na_rm = TRUE, case_weights = NULL, options = list())

roc_aunu_vec(
  truth,
  estimate,
  na_rm = TRUE,
  case_weights = NULL,
  options = list(),
  ...
Arguments

- **data**
  A data.frame containing the columns specified by `truth` and `...`.

- **...**
  A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. There should be as many columns as factor levels of `truth`.

- **truth**
  The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a factor vector.

- **na_rm**
  A logical value indicating whether NA values should be stripped before the computation proceeds.

- **case_weights**
  The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector.

- **options**
  [deprecated]
  No longer supported as of yardstick 1.0.0. If you pass something here it will be ignored with a warning.
  Previously, these were options passed on to pROC::roc(). If you need support for this, use the pROC package directly.

- **estimate**
  A matrix with as many columns as factor levels of `truth`. It is assumed that these are in the same order as the levels of `truth`.

Value

A tibble with columns `.metric`, `.estimator`, and `.estimate` and 1 row of values.
For grouped data frames, the number of rows returned will be the same as the number of groups.
For `roc_aunu_vec()`, a single numeric value (or NA).

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument `event_level` to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

This multiclass method for computing the area under the ROC curve uses the uniform class distribution and is equivalent to `roc_auc(estimator = "macro")`.

Author(s)

Julia Silge
roc_aunu

References


See Also

roc_aunp() for computing the area under the ROC curve of each class against the rest, using the a priori class distribution.

Other class probability metrics: average_precision(), brier_class(), classification_cost(), gain_capture(), mn_log_loss(), pr_auc(), roc_auc(), roc_aunp()

Examples

# Multiclass example

# `obs` is a 4 level factor. The first level is "VF", which is the
# "event of interest" by default in yardstick. See the Relevant Level
# section above.
data(hpc_cv)

# You can use the col1:colN tidyselect syntax
library(dplyr)
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  roc_aunu(obs, VF:L)

# Change the first level of `obs` from "VF" to "M" to alter the
# event of interest. The class probability columns should be supplied
# in the same order as the levels.
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  mutate(obs = relevel(obs, "M")) %>%
  roc_aunu(obs, M, VF:L)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
  roc_aunu(obs, VF:L)

# Vector version
# Supply a matrix of class probabilities
fold1 <- hpc_cv %>%
  filter(Resample == "Fold01")

roc_aunu_vec(
  truth = fold1$obs,
  matrix(
    c(fold1$VF, fold1$F, fold1$M, fold1$L),
    ncol = 4
  )
)
roc_curve() constructs the full ROC curve and returns a tibble. See roc_auc() for the area under the ROC curve.

Usage

roc_curve(data, ...)

## S3 method for class 'data.frame'
roc_curve(
  data,
  truth,
  ..., 
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL,
  options = list()
)

Arguments

data  A data.frame containing the columns specified by truth and ....

... A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.

truth The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

event_level A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.
roc_curve

**options**

[deprecated]

No longer supported as of yardstick 1.0.0. If you pass something here it will be ignored with a warning.

Previously, these were options passed on to pROC::roc(). If you need support for this, use the pROC package directly.

**Details**

roc_curve() computes the sensitivity at every unique value of the probability column (in addition to infinity and minus infinity).

There is a ggplot2::autoplot() method for quickly visualizing the curve. This works for binary and multiclass output, and also works with grouped data (i.e. from resamples). See the examples.

**Value**

A tibble with class roc_df or roc_grouped_df having columns .threshold, specificity, and sensitivity.

**Multiclass**

If a multiclass truth column is provided, a one-vs-all approach will be taken to calculate multiple curves, one per level. In this case, there will be an additional column, .level, identifying the "one" column in the one-vs-all calculation.

**Relevant Level**

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

**Author(s)**

Max Kuhn

**See Also**

Compute the area under the ROC curve with roc_auc().

Other curve metrics: gain_curve(), lift_curve(), pr_curve()

**Examples**

```r
# Two class example

# 'truth' is a 2 level factor. The first level is "Class1", which is the
# "event of interest" by default in yardstick. See the Relevant Level
# section above.
```
data(two_class_example)

# Binary metrics using class probabilities take a factor `truth` column, # and a single class probability column containing the probabilities of # the event of interest. Here, since "Class1" is the first level of # "truth", it is the event of interest and we pass in probabilities for it.
roc_curve(two_class_example, truth, Class1)

# Visualize the curve using ggplot2 manually
library(ggplot2)
library(dplyr)
roc_curve(two_class_example, truth, Class1) %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity)) +
  geom_path() +
  geom_abline(lty = 3) +
  coord_equal() +
  theme_bw()

# Or use autoplot
autoplot(roc_curve(two_class_example, truth, Class1))

## Not run:
# Multiclass one-vs-all approach
# One curve per level
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  roc_curve(obs, VF:L) %>%
  autoplot()

# Same as above, but will all of the resamples
hpc_cv %>%
  group_by(Resample) %>%
  roc_curve(obs, VF:L) %>%
  autoplot()

## End(Not run)

---

rpd

### Ratio of performance to deviation

#### Description

These functions are appropriate for cases where the model outcome is a numeric. The ratio of performance to deviation (rpd()) and the ratio of performance to inter-quartile (rpiq()) are both measures of consistency/correlation between observed and predicted values (and not of accuracy).
Usage

rpd(data, ...)

## S3 method for class 'data.frame'
rpd(data, truth, estimate, na.rm = TRUE, case_weights = NULL, ...)
rpd_vec(truth, estimate, na.rm = TRUE, case_weights = NULL, ...)

Arguments

data A data.frame containing the columns specified by the truth and estimate arguments.
...
truth The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
na.rm A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Details

In the field of spectroscopy in particular, the ratio of performance to deviation (RPD) has been used as the standard way to report the quality of a model. It is the ratio between the standard deviation of a variable and the standard error of prediction of that variable by a given model. However, its systematic use has been criticized by several authors, since using the standard deviation to represent the spread of a variable can be misleading on skewed dataset. The ratio of performance to interquartile has been introduced by Bellon-Maurel et al. (2010) to address some of these issues, and generalise the RPD to non-normally distributed variables.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.
For grouped data frames, the number of rows returned will be the same as the number of groups.
For rpd_vec(), a single numeric value (or NA).

Author(s)

Pierre Roudier
rpd

References


See Also

The closely related inter-quartile metric: rpiq()

Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), rpiq(), rsq_trad(), rsq(), smape()

Other consistency metrics: ccc(), rpiq(), rsq_trad(), rsq()

Examples

# Supply truth and predictions as bare column names
rpd(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
  rpd(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))
rpiq  

**Description**

These functions are appropriate for cases where the model outcome is a numeric. The ratio of performance to deviation (rpd()) and the ratio of performance to inter-quartile (rpiq()) are both measures of consistency/correlation between observed and predicted values (and not of accuracy).

**Usage**

rpiq(data, ...)

```r
## S3 method for class 'data.frame'
rpiq(data, truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
```

rpiq_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)

**Arguments**

- **data**: A data.frame containing the columns specified by the truth and estimate arguments.
- **...**: Not currently used.
- **truth**: The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
- **estimate**: The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
- **na_rm**: A logical value indicating whether NA values should be stripped before the computation proceeds.
- **case_weights**: The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

**Details**

In the field of spectroscopy in particular, the ratio of performance to deviation (RPD) has been used as the standard way to report the quality of a model. It is the ratio between the standard deviation of a variable and the standard error of prediction of that variable by a given model. However, its systematic use has been criticized by several authors, since using the standard deviation to represent the spread of a variable can be misleading on skewed dataset. The ratio of performance to inter-quartile has been introduced by Bellon-Maurel et al. (2010) to address some of these issues, and generalise the RPD to non-normally distributed variables.
Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.
For grouped data frames, the number of rows returned will be the same as the number of groups.
For rpd_vec(), a single numeric value (or NA).

Author(s)

Pierre Roudier

References


See Also

The closely related deviation metric: rpd()
Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), rpd(), rsq_trad(), rsq(), smape()
Other consistency metrics: ccc(), rpd(), rsq_trad(), rsq()

Examples

# Supply truth and predictions as bare column names
rpd(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
rsq

rsq(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))

rsq | $R$ squared
--- | ---

Description

Calculate the coefficient of determination using correlation. For the traditional measure of R squared, see `rsq_trad()`.

Usage

rsq(data, ...)

## S3 method for class 'data.frame'
rsq(data, truth, estimate, na_rm = TRUE, case_weights = NULL, ...)

rsq_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)

Arguments

data A `data.frame` containing the columns specified by the truth and estimate arguments.

... Not currently used.

truth The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a numeric vector.

estimate The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a numeric vector.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For `_vec()` functions, a numeric vector.
Details

The two estimates for the coefficient of determination, \( \text{rsq()} \) and \( \text{rsq\_trad()} \), differ by their formula. The former guarantees a value on (0, 1) while the latter can generate inaccurate values when the model is non-informative (see the examples). Both are measures of consistency/correlation and not of accuracy.

\( \text{rsq()} \) is simply the squared correlation between truth and estimate.

Because \( \text{rsq()} \) internally computes a correlation, if either truth or estimate are constant it can result in a divide by zero error. In these cases, a warning is thrown and \( \text{NA} \) is returned. This can occur when a model predicts a single value for all samples. For example, a regularized model that eliminates all predictors except for the intercept would do this. Another example would be a CART model that contains no splits.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For \( \text{rsq\_vec()} \), a single numeric value (or \( \text{NA} \)).

Author(s)

Max Kuhn

References


See Also

Other numeric metrics: \( \text{ccc()} \), \( \text{huber\_loss\_pseudo()} \), \( \text{huber\_loss()} \), \( \text{iic()} \), \( \text{mae()} \), \( \text{mape()} \), \( \text{mase()} \), \( \text{mpe()} \), \( \text{msd()} \), \( \text{poisson\_log\_loss()} \), \( \text{rmse()} \), \( \text{rpd()} \), \( \text{rpiq()} \), \( \text{rsq\_trad()} \), \( \text{smape()} \)

Other consistency metrics: \( \text{ccc()} \), \( \text{rpd()} \), \( \text{rpiq()} \), \( \text{rsq\_trad()} \)

Examples

# Supply truth and predictions as bare column names
\( \text{rsq(solubility\_test, solubility, prediction)} \)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
\( \text{solubility\_resampled <- bind\_rows(}\)
  \( \text{replicate(} \)
    \( \text{n = times,} \)
    \( \text{expr = sample\_n(solubility\_test, size, replace = TRUE),} \)
    \( \text{simplify = FALSE} \)
  \( \text{)} \)

\( \text{)} \)

\( \text{)} \)
### rsq_trad

**R squared - traditional**

**Description**

Calculate the coefficient of determination using the traditional definition of R squared using sum of squares. For a measure of R squared that is strictly between (0, 1), see `rsq()`.

**Usage**

```r
rsq_trad(data, ...)
```

```r
## S3 method for class 'data.frame'
rsq_trad(data, truth, estimate, na.rm = TRUE, case_weights = NULL, ...)
```

```r
rsq_trad_vec(truth, estimate, na.rm = TRUE, case_weights = NULL, ...)
```

**Arguments**

- `data` A data.frame containing the columns specified by the truth and estimate arguments.
... Not currently used.

truth The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.

estimate The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.

na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.

case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Details

The two estimates for the coefficient of determination, rsq() and rsq_trad(), differ by their formula. The former guarantees a value on (0, 1) while the latter can generate inaccurate values when the model is non-informative (see the examples). Both are measures of consistency/correlation and not of accuracy.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For rsq_trad_vec(), a single numeric value (or NA).

Author(s)

Max Kuhn

References


See Also

Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), rpd(), rpiq(), rsq(), smape()

Other consistency metrics: ccc(), rpd(), rpiq(), rsq()

Examples

# Supply truth and predictions as bare column names
rsq_trad(solubility_test, solubility, prediction)

library(dplyr)
set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
  rsq_trad(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))

# With uninformative data, the traditional version of R^2 can return
# negative values.
set.seed(2291)
solubility_test$randomized <- sample(solubility_test$prediction)
rsq(solubility_test, solubility, randomized)
rsq_trad(solubility_test, solubility, randomized)

---

### sens

<table>
<thead>
<tr>
<th>Sensitivity</th>
</tr>
</thead>
</table>

**Description**

These functions calculate the `sens()` (sensitivity) of a measurement system compared to a reference result (the "truth" or gold standard). Highly related functions are `spec()`, `ppv()`, and `npv()`.

**Usage**

```
sens(data, ...)
```

## S3 method for class 'data.frame'
sens(
  data,
  truth,
  estimate,
```
estimator = NULL,
na_rm = TRUE,
case_weights = NULL,
event_level = yardstick_event_level(),
...)

sens_vec(
  truth,
estimate,
estimator = NULL,
na_rm = TRUE,
case_weights = NULL,
event_level = yardstick_event_level(),
...)

sensitivity(data, ...)

## S3 method for class 'data.frame'
sensitivity(
  data,
  truth,
estimate,
estimator = NULL,
na_rm = TRUE,
case_weights = NULL,
event_level = yardstick_event_level(),
...)

sensitivity_vec(
  truth,
estimate,
estimator = NULL,
na_rm = TRUE,
case_weights = NULL,
event_level = yardstick_event_level(),
...)

Arguments

| data | Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table. |
| ... | Not currently used. |
| truth | The column identifier for the true class results (that is a factor). This should be |
an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.

**estimate**

The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

**estimator**

One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

**na_rm**

A logical value indicating whether NA values should be stripped before the computation proceeds.

**case_weights**

The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

**event_level**

A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.

**Details**

The sensitivity (sens()) is defined as the proportion of positive results out of the number of samples which were actually positive.

When the denominator of the calculation is 0, sensitivity is undefined. This happens when both # true_positive = 0 and # false_negative = 0 are true, which mean that there were no true events. When computing binary sensitivity, a NA value will be returned with a warning. When computing multiclass sensitivity, the individual NA values will be removed, and the computation will proceed, with a warning.

**Value**

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For sens_vec(), a single numeric value (or NA).

**Relevant Level**

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.
Multiclass

Macro, micro, and macro-weighted averaging is available for this metric. The default is to select macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary calculation is done. See vignette("multiclass", "yardstick") for more information.

Implementation

Suppose a 2x2 table with notation:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Predicted</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>C</td>
<td>D</td>
<td></td>
</tr>
</tbody>
</table>

The formulas used here are:

\[
\text{Sensitivity} = \frac{A}{A + C}
\]

\[
\text{Specificity} = \frac{D}{B + D}
\]

\[
\text{Prevalence} = \frac{(A + C)}{(A + B + C + D)}
\]

\[
PPV = \frac{(\text{Sensitivity} \times \text{Prevalence})}{((\text{Sensitivity} \times \text{Prevalence}) + ((1 - \text{Specificity}) \times (1 - \text{Prevalence})))}
\]

\[
NPV = \frac{(\text{Specificity} \times (1 - \text{Prevalence}))}{(((1 - \text{Sensitivity}) \times \text{Prevalence}) + ((\text{Specificity} \times (1 - \text{Prevalence}))}
\]

See the references for discussions of the statistics.

Author(s)

Max Kuhn

References


See Also

Other class metrics: accuracy(), bal_accuracy(), detection_prevalence(), f_meas(), j_index(), kap(), mcc(), npv(), ppv(), precision(), recall(), spec()

Other sensitivity metrics: npv(), ppv(), spec()
Examples

# Two class
data("two_class_example")
sens(two_class_example, truth, predicted)

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
sens(obs, pred)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
sens(obs, pred)

# Weighted macro averaging
hpc_cv %>%
  group_by(Resample) %>%
sens(obs, pred, estimator = "macro_weighted")

# Vector version
sens_vec(
  two_class_example$truth,
  two_class_example$predicted
)

# Making Class2 the "relevant" level
sens_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)

_________________________________________________________________________

smape Symmetric mean absolute percentage error

_________________________________________________________________________

Description

Calculate the symmetric mean absolute percentage error. This metric is in relative units.

Usage

smape(data, ...)

## S3 method for class 'data.frame'
smape(data, truth, estimate, na_rm = TRUE, case_weights = NULL, ...)
smape_vec(truth, estimate, na_rm = TRUE, case_weights = NULL, ...)

Arguments

data  A data.frame containing the columns specified by the truth and estimate arguments.
...
truth  The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate  The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
na_rm  A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights  The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.

Details

This implementation of smape() is the "usual definition" where the denominator is divided by two.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.
For grouped data frames, the number of rows returned will be the same as the number of groups.
For smape_vec(), a single numeric value (or NA).

Author(s)

Max Kuhn, Riaz Hedayati

See Also

Other numeric metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), rpd(), rpiq(), rsq_trad(), rsq()

Other accuracy metrics: ccc(), huber_loss_pseudo(), huber_loss(), iic(), mae(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse()
Examples

# Supply truth and predictions as bare column names
smape(solubility_test, solubility, prediction)

library(dplyr)

set.seed(1234)
size <- 100
times <- 10

# create 10 resamples
solubility_resampled <- bind_rows(
  replicate(
    n = times,
    expr = sample_n(solubility_test, size, replace = TRUE),
    simplify = FALSE
  ),
  .id = "resample"
)

# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
  smape(solubility, prediction)

metric_results

# Resampled mean estimate
metric_results %>%
  summarise(avg_estimate = mean(.estimate))

solubility_test

Solubility Predictions from MARS Model

Description

Solubility Predictions from MARS Model

Details

For the solubility data in Kuhn and Johnson (2013), these data are the test set results for the MARS model. The observed solubility (in column `solubility`) and the model results (`prediction`) are contained in the data.

Value

solubility_test

a data frame
Source

Examples
```
data(solubility_test)
str(solubility_test)
```

---

**spec**

**Specificity**

These functions calculate the `spec()` (specificity) of a measurement system compared to a reference result (the "truth" or gold standard). Highly related functions are `sens()`, `ppv()`, and `npv()`.

**Usage**

```r
spec(data, ...)
```

## S3 method for class 'data.frame'

```r
spec(
  data,
  truth,
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)
```

```r
spec_vec(
  truth,
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)
```

```r
specificity(data, ...)
```

## S3 method for class 'data.frame'

```r
specificity(
  data,
```
spec

truth,
estimate,
estimator = NULL,
na_rm = TRUE,
case_weights = NULL,
event_level = yardstick_event_level(),
...
)

specificity_vec(
  truth,
estimate,
estimator = NULL,
na_rm = TRUE,
case_weights = NULL,
event_level = yardstick_event_level(),
...
)

Arguments

data Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
...
Not currently used.
truth The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector.
event_level A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first", however, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.
Details

The specificity measures the proportion of negatives that are correctly identified as negatives.

When the denominator of the calculation is 0, specificity is undefined. This happens when both
\# true_negative = 0 and \# false_positive = 0 are true, which mean that there were no true
negatives. When computing binary specificity, a NA value will be returned with a warning. When
computing multiclass specificity, the individual NA values will be removed, and the computation
will proceed, with a warning.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values.

For grouped data frames, the number of rows returned will be the same as the number of groups.

For spec_vec(), a single numeric value (or NA).

Relevant Level

There is no common convention on which factor level should automatically be considered the
"event" or "positive" result when computing binary classification metrics. In yardstick, the default
is to use the first level. To alter this, change the argument event_level to "second" to consider
the last level of the factor the level of interest. For multiclass extensions involving one-vs-all com-
parisons (such as macro averaging), this option is ignored and the "one" level is always the relevant
result.

Multiclass

Macro, micro, and macro-weighted averaging is available for this metric. The default is to select
macro averaging if a truth factor with more than 2 levels is provided. Otherwise, a standard binary
calculation is done. See vignette("multiclass", "yardstick") for more information.

Implementation

Suppose a 2x2 table with notation:

<table>
<thead>
<tr>
<th>Reference</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Positive</td>
<td>Negative</td>
</tr>
<tr>
<td>Predicted</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>Negative</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

The formulas used here are:

\[
\text{Sensitivity} = \frac{A}{A + C} \\
\text{Specificity} = \frac{D}{B + D} \\
\text{Prevalence} = \frac{A + C}{A + B + C + D} \\
\text{PPV} = \frac{(\text{Sensitivity} \times \text{Prevalence})}{(((\text{Sensitivity} \times \text{Prevalence}) + \{(1 - \text{Specificity}) \times (1 - \text{Prevalence})\})} \\
\text{NPV} = \frac{(\text{Specificity} \times (1 - \text{Prevalence}))}{(((1 - \text{Sensitivity}) \times \text{Prevalence}) + (\text{Specificity} \times (1 - \text{Prevalence})))}
\]
See the references for discussions of the statistics.

Author(s)
Max Kuhn

References

See Also
Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `j_index()`, `kap()`, `mcc()`, `npv()`, `ppv()`, `precision()`, `recall()`, `sens()`

Other sensitivity metrics: `npv()`, `ppv()`, `sens()`

Examples
# Two class
data("two_class_example")
spec(two_class_example, truth, predicted)

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv %>%
  filter(Resample == "Fold01") %>%
spec(obs, pred)

# Groups are respected
hpc_cv %>%
  group_by(Resample) %>%
spec(obs, pred)

# Weighted macro averaging
hpc_cv %>%
  group_by(Resample) %>%
spec(obs, pred, estimator = "macro_weighted")

# Vector version
spec_vec(
  two_class_example$truth,
  two_class_example$predicted
)

# Making Class2 the "relevant" level
spec_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
summary.conf_mat

Summary Statistics for Confusion Matrices

Description

Various statistical summaries of confusion matrices are produced and returned in a tibble. These include those shown in the help pages for sens(), recall(), and accuracy(), among others.

Usage

## S3 method for class 'conf_mat'
summary(
  object,
  prevalence = NULL,
  beta = 1,
  estimator = NULL,
  event_level = yardstick_event_level(),
  ...
)

Arguments

- **object**: An object of class conf_mat().
- **prevalence**: A number in (0, 1) for the prevalence (i.e. prior) of the event. If left to the default, the data are used to derive this value.
- **beta**: A numeric value used to weight precision and recall for f_meas().
- **estimator**: One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
- **event_level**: A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that generally defaults to "first". However, if the deprecated global option yardstick.event_first is set, that will be used instead with a warning.
- **...**: Not currently used.

Value

A tibble containing various classification metrics.
Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the first level. To alter this, change the argument event_level to "second" to consider the last level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

See Also

conf_mat()

Examples

data("two_class_example")

cmat <- conf_mat(two_class_example, truth = "truth", estimate = "predicted")
summary(cmat)
summary(cmat, prevalence = 0.70)

library(dplyr)
library(tidyr)
data("hpc_cv")

# Compute statistics per resample then summarize
all_metrics <- hpc_cv %>%
  group_by(Resample) %>%
  conf_mat(obs, pred) %>%
  mutate(summary_tbl = lapply(conf_mat, summary)) %>%
  unnest(summary_tbl)

all_metrics %>%
  group_by(.metric) %>%
  summarise(
    mean = mean(.estimate, na.rm = TRUE),
    sd = sd(.estimate, na.rm = TRUE)
  )

---

two_class_example   Two Class Predictions

Description

Two Class Predictions

Details

These data are a test set form a model built for two classes ("Class1" and "Class2"). There are columns for the true and predicted classes and column for the probabilities for each class.
yardstick_remove_missing

**Value**

two_class_example

  a data frame

**Examples**

data(two_class_example)
str(two_class_example)

# `truth` is a 2 level factor. The first level is "Class1", which is the
# "event of interest" by default in yardstick. See the Relevant Level
# section in any classification function (such as `?pr_auc`) to see how
# to change this.
levels(hpc_cv$obs)

---

**yardstick_remove_missing**

*Developer function for handling missing values in new metrics*

**Description**

`yardstick_remove_missing()`, and `yardstick_any_missing()` are useful alongside the metric-summarizers functions for implementing new custom metrics. `yardstick_remove_missing()` removes any observations that contains missing values across, truth, estimate and case_weights. `yardstick_any_missing()` returns FALSE if there is any missing values in the inputs.

**Usage**

`yardstick_remove_missing(truth, estimate, case_weights)`

`yardstick_any_missing(truth, estimate, case_weights)`

**Arguments**

- `truth`, `estimate`  
  Vectors of the same length.

- `case_weights`  
  A vector of the same length as truth and estimate, or NULL if case weights are not being used.

**See Also**

metric-summarizers
Index

* accuracy metrics
  - ccc, 12
  - huber_loss, 38
  - huber_loss_pseudo, 40
  - iic, 42
  - mae, 53
  - mape, 55
  - mase, 56
  - mpe, 72
  - msd, 74
  - poisson_log_loss, 81
  - rmse, 99
  - smape, 125

* class metrics
  - accuracy, 3
  - bal_accuracy, 8
  - detection_prevalence, 22
  - f_meas, 27
  - j_index, 44
  - kap, 47
  - mcc, 59
  - npv, 77
  - ppv, 83
  - precision, 86
  - recall, 95
  - sens, 121
  - spec, 128

* class probability metrics
  - average_precision, 4
  - brier_class, 10
  - classification_cost, 16
  - gain_capture, 30
  - mn_log_loss, 70
  - pr_auc, 90
  - roc_auc, 101
  - roc_aunp, 104
  - roc_aunu, 107

* consistency metrics
  - ccc, 12
  - rpd, 112
  - rpiq, 115
  - rsq, 117
  - rsq_trad, 119

* curve metrics
  - gain_curve, 34
  - lift_curve, 49
  - pr_curve, 93
  - roc_curve, 110

* datasets
  - hpc_cv, 37
  - lung_surv, 52
  - pathology, 81
  - solubility_test, 127
  - two_class_example, 133

* numeric metrics
  - ccc, 12
  - huber_loss, 38
  - huber_loss_pseudo, 40
  - iic, 42
  - mae, 53
  - mape, 55
  - mase, 56
  - mpe, 72
  - msd, 74
  - poisson_log_loss, 81
  - rmse, 99
  - rpd, 112
  - rpiq, 115
  - rsq, 117
  - rsq_trad, 119
  - smape, 125

* relevance metrics
  - f_meas, 27
  - precision, 86
  - recall, 95

* sensitivity metrics
  - npv, 77
  - ppv, 83
sens, 121
spec, 128
abort(), 16, 25, 63
accuracy, 3, 10, 23, 29, 46, 49, 60, 80, 85, 89, 98, 124, 131
accuracy(), 47, 65, 71, 132
accuracy_vec(accuracy), 3
average_precision, 4, 12, 18, 32, 71, 91, 103, 106, 109
average_precision_vec
(average_precision), 4
bal_accuracy, 4, 8, 23, 29, 46, 49, 60, 80, 85, 89, 98, 124, 131
bal_accuracy_vec(bal_accuracy), 8
base::table(), 20
brier_class, 7, 10, 18, 32, 71, 91, 103, 106, 109
brier_class_vec(brier_class), 10
ccc, 12, 39, 41, 43, 54, 56, 58, 73, 76, 82, 100, 114, 116, 118, 120, 126
ccc(), 13
ccc_vec(ccc), 12
check_class_metric(check_metric), 15
class_metric_summarizer
(metric-summarizers), 61
classification_cost, 7, 12, 16, 32, 71, 91, 103, 106, 109
classification_cost_vec
(classification_cost), 16
conf_mat, 19
conf_mat(), 20, 132, 133
curve_metric_summarizer
(metric-summarizers), 61
curve_survival_metric_summarizer
(metric-summarizers), 61
detection_prevalence, 4, 10, 22, 29, 46, 49, 60, 80, 85, 89, 98, 124, 131
detection_prevalence_vec
(detection_prevalence), 22
developer helpers, 24
dots_to_estimate(developer helpers), 24
dots_to_estimate(), 63, 64
dynamic_survival_metric_summarizer
(metric-summarizers), 61
f_meas, 4, 10, 23, 27, 46, 49, 60, 80, 85, 89, 98, 124, 131
f_meas(), 27, 69, 86, 95, 132
f_meas_vec(f_meas), 27
finalize_estimator(developer helpers), 24
finalize_estimator(), 64
finalize_estimator_internal
(developer helpers), 24
gain_capture, 7, 12, 18, 30, 71, 91, 103, 106, 109
gain_capture(), 34, 36
gain_capture_vec(gain_capture), 30
gain_curve, 34, 52, 94, 111
gain_curve(), 32, 49
get_weights(developer helpers), 24
ggplot2::autoplot(), 20, 35, 50, 94, 111
hpc_cv, 37
huber_loss, 14, 38, 41, 43, 54, 56, 58, 73, 76, 82, 100, 114, 116, 118, 120, 126
huber_loss(), 40
huber_loss_pseudo, 14, 39, 40, 43, 54, 56, 58, 73, 76, 82, 100, 114, 116, 118, 120, 126
huber_loss_pseudo_vec
(huber_loss_pseudo), 40
huber_loss_vec(huber_loss), 38
iic, 14, 39, 41, 42, 54, 56, 58, 73, 76, 82, 100, 114, 116, 118, 120, 126
iic_vec(iic), 42
j_index, 4, 10, 23, 29, 44, 49, 60, 80, 85, 89, 98, 124, 131
j_index_vec(j_index), 44
kap, 4, 10, 23, 29, 46, 47, 60, 80, 85, 89, 98, 124, 131
kap(), 65
kap_vec(kap), 47
lift_curve, 36, 49, 94, 111
lift_curve(), 34
lung, 53
lung_surv, 52

mae, 14, 39, 41, 43, 53, 56, 58, 73, 76, 82, 100, 114, 116, 118, 120, 126
mae(), 65, 74, 75
mae_vec (mae), 53
mape, 14, 39, 41, 43, 54, 55, 58, 73, 76, 82, 100, 114, 116, 118, 120, 126
mape_vec (mape), 55
mase, 14, 39, 41, 43, 54, 56, 73, 76, 82, 100, 114, 116, 118, 120, 126
mase_vec (mase), 56
mcc, 4, 10, 23, 29, 46, 49, 59, 80, 85, 89, 98, 124, 131
mcc_vec (mcc), 59
metric_summarizers, 24
metric_summarizers, 15, 16, 26, 61, 134
metric_set, 66
metric_set(), 65, 68, 76
metric_tweak, 68
metrics, 64
metrics(), 67
mn_log_loss, 7, 12, 18, 32, 70, 91, 103, 106, 109
mn_log_loss(), 65
mn_log_loss_vec (mn_log_loss), 70
mpe, 14, 39, 41, 43, 54, 56, 58, 72, 76, 82, 100, 114, 116, 118, 120, 126
mpe_vec (mpe), 72
msd, 14, 39, 41, 43, 54, 56, 58, 73, 74, 82, 100, 114, 116, 118, 120, 126
msd_vec (msd), 74

new_metric, 76
new_class_metric (new_metric), 76
new_dynamic_survival_metric (new_metric), 76
new_integrated_survival_metric (new_metric), 76
new_numeric_metric (new_metric), 76
new_prob_metric (new_metric), 76
new_static_survival_metric (new_metric), 76

npv, 4, 10, 23, 29, 46, 49, 60, 77, 85, 89, 98, 124, 131
npv(), 77, 78, 83, 84, 121, 128
npv_vec (npv), 77

numeric_metric_summarizer (metric_summarizers), 61

pathology, 81
poisson_log_loss, 14, 39, 41, 43, 54, 56, 58, 73, 76, 81, 100, 114, 116, 118, 120, 126
poisson_log_loss_vec (poisson_log_loss), 81
ppv, 4, 10, 23, 29, 46, 49, 60, 80, 83, 89, 98, 124, 131
ppv(), 77, 78, 83, 84, 121, 128
ppv_vec (ppv), 83
pr_auc, 7, 12, 18, 32, 71, 90, 103, 106, 109
pr_auc(), 7, 93, 94
pr_auc_vec (pr_auc), 90
pr_curve, 36, 52, 93, 111
pr_curve(), 4, 6, 7, 90, 91
precision, 4, 10, 23, 29, 46, 49, 60, 80, 85, 86, 98, 124, 131
precision(), 27, 86, 95
precision_vec (precision), 86
prob_metric_summarizer (metric_summarizers), 61


recall, 4, 10, 23, 29, 46, 49, 60, 80, 85, 89, 95, 124, 131
recall(), 27, 86, 95, 132
recall_vec (recall), 95
rmse, 14, 39, 41, 43, 54, 56, 58, 73, 76, 82, 99, 114, 116, 118, 120, 126
rmse(), 13, 38, 40, 65
rmse_vec (rmse), 99
roc_auc, 7, 12, 18, 32, 71, 91, 101, 106, 109
roc_auc(), 65, 110, 111
roc_auc_vec (roc_auc), 101
roc_aunp, 7, 12, 18, 32, 71, 91, 103, 104, 109
roc_aunp(), 109
roc_aunp_vec (roc_aunp), 104
roc_aunu, 7, 12, 18, 32, 71, 91, 103, 106, 107
roc_aunu(), 106
roc_aunu_vec (roc_aunu), 107
roc_curve, 36, 52, 94, 110
roc_curve(), 101, 103
rpd, 14, 39, 41, 43, 54, 56, 58, 73, 76, 82, 100,
112, 116, 118, 120, 126
rpd(), 112, 115, 116
rpd_vec (rpd), 112
rpiq, 14, 39, 41, 43, 54, 56, 58, 73, 76, 82,
100, 114, 115, 118, 120, 126
rpiq(), 112, 114, 115
rpiq_vec (rpiq), 115
rsq, 14, 39, 41, 43, 54, 56, 58, 73, 76, 82, 100,
114, 116, 117, 120, 126
rsq(), 13, 65, 118–120
rsq_trad, 14, 39, 41, 43, 54, 56, 58, 73, 76,
82, 100, 114, 116, 118, 119, 126
rsq_trad(), 117, 118, 120
rsq_trad_vec (rsq_trad), 119
rsq_vec (rsq), 117
sens, 4, 10, 23, 29, 46, 49, 60, 80, 85, 89, 98,
121, 131
sens(), 8, 44, 77, 83, 121, 128, 132
sens_vec (sens), 121
sensitivity (sens), 121
sensitivity_vec (sens), 121
smape, 14, 39, 41, 43, 54, 56, 58, 73, 76, 82,
100, 114, 116, 118, 120, 125
smape_vec (smape), 125
solubility.test, 127
spec, 4, 10, 23, 29, 46, 49, 60, 80, 85, 89, 98,
124, 128
spec(), 8, 44, 77, 83, 121, 128
spec_vec (spec), 128
specificity (spec), 128
specificity_vec (spec), 128
static_survival.metric.summarizer
(metric-summarizers), 61
summary.conf_mat, 132
summary.conf_mat(), 20, 21
tidy.conf_mat (conf_mat), 19
two_class.example, 133
validate.estimator (developer-helpers),
24
yardstick.any.missing
(yardstick.remove.missing), 134
yardstick.remove.missing, 24, 26, 61, 64,
134