Package ‘yardstick’
October 14, 2022

Type Package
Title Tidy Characterizations of Model Performance
Version 1.1.0
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 dplyr (>= 1.0.9), generics (>= 0.1.2), hardhat (>= 1.0.0), rlang (>= 1.0.2), tidyselect (>= 1.1.2), utils, vctrs (>= 0.4.1)
Suggests covr, crayon, ggplot2, knitr, probably (>= 0.0.6), rmarkdown, testthat (>= 3.0.0), tidyr
VignetteBuilder knitr
Config/Needs/website tidyverse/tidytemplate
Config/testthat/edition 3
Encoding UTF-8
LazyData true
RoxygenNote 7.2.1
NeedsCompilation yes
Author Max Kuhn [aut], Davis Vaughan [aut], Emil Hvitfeldt [aut, cre] (https://orcid.org/0000-0002-0679-1945), RStudio [cph, fnd]
Maintainer Emil Hvitfeldt <emil.hvitfeldt@rstudio.com>
Repository CRAN
Date/Publication 2022-09-07 22:30:02 UTC
R topics documented:

accuracy ................................................................. 3
average_precision ....................................................... 4
bal_accuracy ............................................................. 8
ccc ................................................................. 10
classification_cost ..................................................... 13
conf_mat .............................................................. 16
detection_prevalence .................................................. 18
f_meas .............................................................. 21
gain_capture .......................................................... 24
gain_curve ............................................................ 28
get_weights .............................................................. 31
hpc_cv .............................................................. 33
huber_loss ........................................................... 34
huber_loss_pseudo ..................................................... 36
iic .............................................................. 38
j_index ............................................................. 40
kap ............................................................. 43
lift_curve ............................................................. 46
mae ............................................................. 49
mape ............................................................. 50
mase ............................................................. 52
mcc ............................................................. 55
metrics ............................................................. 57
metric_set ............................................................. 58
metric_summarizer .................................................... 61
metric_tweak .......................................................... 62
metric_vec_template ................................................... 63
mn_log_loss ............................................................ 65
mpe ............................................................. 67
msd ............................................................. 69
new-metric ............................................................. 71
npv ............................................................. 72
pathology ............................................................ 75
poisson_log_loss ....................................................... 76
ppv ............................................................. 77
precision ............................................................ 81
pr_auc ............................................................ 84
pr_curve ............................................................. 87
recall ............................................................. 90
rmse ............................................................. 93
roc_auc ............................................................ 95
roc_aunp ............................................................ 99
roc_aunu ........................................................... 101
roc_curve ........................................................... 104
rpd ............................................................. 106
rpiq ............................................................. 109
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.
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. Otherwise, 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.

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.
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).

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

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.
average_precision

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: classification_cost(), gain_capture(), 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.
average_precision(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") %>%
  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 %>%
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.
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")
```

Concordance correlation coefficient
Description

Calculate the concordance correlation coefficient.

Usage

```r
ccc(data, ...)
## S3 method for class 'data.frame'
ccc(
  data,
  truth,
  estimate,
  bias = FALSE,
  na_rm = TRUE,
  case_weights = NULL,
  ...
)
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(), mape(), mase(), mpe(), msd(), poisson_log_loss(), rmse(), rpd(), rpiq(), rsq(), rsq_trad(), smape()

Other consistency metrics: rpd(), rpiq(), rsq(), rsq_trad()

Other accuracy metrics: huber_loss_pseudo(), huber_loss(), iic(), 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)
classification_cost

metric_results

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

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,
  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. Otherwise, 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.

costs  A data frame with columns "truth", "estimate", and "cost".
"truth" and "estimate" should be character columns containing unique combi-
nations 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 * 0 +
0.3 * 5 + 0.4 * 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
classification_cost

See Also

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

Examples

library(dplyr)

# Two class example
data(two_class_example)

# Assuming 'Class1' is our "event", this penalizes false positives heavily
costs1 <- tribble(~truth, ~estimate, ~cost,
                  "Class1", "Class2", 1,
                  "Class2", "Class1", 2)

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

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

# 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,
# 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

conf_mat(data, ...)

## S3 method for class 'data.frame'
conf_mat(
  data,
  truth,
  estimate,
  dnn = c("Prediction", "Truth"),
  case_weights = NULL,
  ...
)

## 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.
conf_mat

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

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)

# 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

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

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.

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

# 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)
f_meas

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

<table>
<thead>
<tr>
<th>f_meas</th>
<th>F Measure</th>
</tr>
</thead>
</table>

Description

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

Usage

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(),
  ...)

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 \textit{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.

\textbf{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.

\textbf{Implementation}

Suppose a 2x2 table with notation:

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

The formulas used here are:

\begin{align*}
\text{recall} &= \frac{A}{(A + C)} \\
\text{precision} &= \frac{A}{(A + B)} \\
F_{\text{meas}} &= (1 + \beta^2) \ast \text{precision} \ast \text{recall}/((\beta^2 \ast \text{precision}) + \text{recall})
\end{align*}

See the references for discussions of the statistics.

\textbf{Author(s)}

Max Kuhn

\textbf{References}


\textbf{See Also}

Other class metrics: \texttt{accuracy()}, \texttt{bal_accuracy()}, \texttt{detection_prevalence()}, \texttt{j_index()}, \texttt{kap()}, \texttt{mcc()}, \texttt{npv()}, \texttt{ppv()}, \texttt{precision()}, \texttt{recall()}, \texttt{sens()}, \texttt{spec()}

Other relevance metrics: \texttt{precision()}, \texttt{recall()}
Examples

# 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

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,  
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. Otherwise, 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.  
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
It is assumed that these are in the same order as the levels of truth.

Details

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

The 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.

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(), 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.
da(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.
da(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
  )
)

# Visualize gain_capture()

# 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))

gain_curve

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, ...)

## 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. Otherwise, there should be as many columns as factor levels of truth.
### Gain Curve

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.

**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.

### 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, .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

**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

gain_curve(two_class_example, truth, Class1)
```
# Base functions

# 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()

---

**get_weights**

*Developer helpers*

### Description

Helpers to be used alongside `metric_vec_template()` and `metric_summarizer()` when creating new metrics. See Custom performance metrics for more information.

### Usage

```r
get_weights(data, estimator)
finalize_estimator(x, estimator = NULL, metric_class = "default")
finalize_estimator_internal(metric_dispatcher, x, estimator)
dots_to_estimate(data, ...)
validate_estimator(estimator, estimator_override = NULL)
```
Arguments

data
A table with truth values as columns and predicted values as rows.
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().
metric_dispatcher
A simple dummy object with the class provided to metric_class. This is created and passed along for you.
... 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. Otherwise, there should be as many columns as factor levels of truth.
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.

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:
Dots -> Estimate

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_summarizer() and evaluate to either a vector or a matrix for further use in the underlying vector functions.

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_summarizer() metric_vec_template()
**Examples**

```r
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

#### Huber loss

**Description**

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

**Usage**

```r
huber_loss(data, ...)

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

huber_loss_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.

**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.

**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**

huber_loss_pseudo(data, ...)

## S3 method for class 'data.frame'

huber_loss_pseudo(  
data,  
truth,  
estimate,  
delta = 1,  
na.rm = TRUE,  
case_weights = NULL,  
...
)

huber_loss_pseudo_vec(  
truth,  
estimate,  
delta = 1,  
na.rm = TRUE,  
...  )
huber_loss_pseudo

    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 sup-
          ports 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.

  delta   A single numeric value. Defines the boundary where the loss function transi-
          tions 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_pseudo_vec(), a single numeric value (or NA).

Author(s)

  James Blair

References


See Also

  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()
Examples

# 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\(^\text{\textsuperscript{i}}\)**

*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.
iic

Usage

iic(data, ...)

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

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.

... 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 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()
Examples

# Supply truth and predictions as bare column names
table(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) %>%
iic(solubility, prediction)

metric_results

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


---

### j_index

#### J-index

**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**

j_index(data, ...)

## S3 method for class 'data.frame'
j_index(
  data,
  truth,
)
j_index

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

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)

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

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

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

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

Kappa

Description

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

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

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.

**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

```r
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

```r
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. Otherwise, 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.

- **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.
- `.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

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) %>%
```
```r
lift_curve(obs, VF:L) %>%
autoplot()
```

### `mae`

**Mean absolute error**

#### Description

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

#### Usage

```r
mae(data, ...)
```

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

```r
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(), smape()

Examples

```r
# 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))
```

<table>
<thead>
<tr>
<th>mape</th>
<th>Mean absolute percent error</th>
</tr>
</thead>
</table>

Description

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

Usage

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(), rpiq(), rsq(), smape() 

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

# 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,
mase

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 computed 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 error 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


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.
- **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 \texttt{mcc\_vec()}, a single numeric value (or \texttt{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 \texttt{yardstick}, the default is to use the \texttt{first} level. To alter this, change the argument \texttt{event\_level} to "second" to consider the \texttt{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

\texttt{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: \texttt{accuracy()}, \texttt{bal\_accuracy()}, \texttt{detection\_prevalence()}, \texttt{f\_meas()}, \texttt{j\_index()}, \texttt{kap()}, \texttt{npv()}, \texttt{ppv()}, \texttt{precision()}, \texttt{recall()}, \texttt{sens()}, \texttt{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 %>%
```r
group_by(Resample) %>%
mcc(obs, pred)
```

---

**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**

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

```r
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. Otherwise, there should be as many columns as 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.  

- **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()`.  

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.

- 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 multinomial 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)
Details

All functions must be either:

- Only numeric metrics
- A mix of class metrics or class prob 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.

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

```r
# Class / prob metric set signature:
fn(
  data,
  truth,
  ...
  estimate,
  estimator = NULL,
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  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 ....

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)
### `metric_summarizer` Developer function for summarizing new metrics

**Description**

`metric_summarizer()` is useful alongside `metric_vec_template()` for implementing new custom metrics. `metric_summarizer()` calls the metric function inside `dplyr::summarise()`. `metric_vec_template()` is a generalized function that calls the core implementation of a metric function, and includes a number of checks on the types, lengths, and argument inputs. See Custom performance metrics for more information.

**Usage**

```r
metric_summarizer(
  metric_nm, metric_fn, data, truth, estimate, estimator = NULL, na_rm = TRUE, event_level = NULL, case_weights = NULL, ...
  metric_fn_options = list()
)
```

**Arguments**

- **metric_nm** 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.
- **metric_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 `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()`.
- **estimator** For numeric metrics, this is left as NULL so averaging is not passed on to the metric function implementation. For classification metrics, 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.
- **na_rm** A logical value indicating whether NA values should be stripped before the computation proceeds. The removal is executed in `metric_vec_template()`.
event_level For numeric metrics, this is left as NULL to prevent it from being passed on to the metric function implementation. For classification metrics, this can either be NULL to use the default event_level value of the metric_fn or a single string of either "first" or "second" to pass along describing which level should be considered the "event".

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 metric_fn as the named argument case_weights.

... Currently not used. Metric specific options are passed in through metric_fn_options.

metric_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.

Details

metric_summarizer() is 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

metric_vec_template() finalize_estimator() dots_to_estimate()

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 f_meas()). Using metric_tweak(), you can set optional arguments to custom values ahead of time, before they go into the metric set.

Usage

metric_tweak(.name, .fn, ...)
Arguments

- **.name**: A single string giving the name of the new metric. This will be used in the ".metric" column of the output.
- **.fn**: An existing yardstick metric function to tweak.
- **...**: Name-value pairs specifying which optional arguments to override and the values to replace them with.

Arguments `data`, `truth`, and `estimate` are considered *protected*, and cannot be overridden, but all other optional arguments can be altered.

Details

The function returned from `metric_tweak()` only takes ... as arguments, which are passed through to the original `.fn`. Passing `data`, `truth`, and `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.

Value

A tweaked version of `.fn`, updated to use new defaults supplied in ....

Examples

```r
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)
m airlite <- metric_tweak("airlit", airlite, m = 5)
metrics <- metric_set(mase, mase10, mase12)
m airlite <- metric_vec_template
metrics(solubility_test, solubility, prediction)
```

Description

`metric_vec_template()` is useful alongside `metric_summarizer()` for implementing new custom metrics. `metric_summarizer()` calls the metric function inside `dplyr::summarise()`. `metric_vec_template()` is a generalized function that calls the core implementation of a metric function, and includes a number of checks on the types, lengths, and argument inputs.
Usage

metric_vec_template(
  metric_impl,
  truth,
  estimate,
  na_rm = TRUE,
  cls = "numeric",
  estimator = NULL,
  case_weights = NULL,
  ...
)

Arguments

metric_impl  The core implementation function of your custom metric. This core implementation function is generally defined inside the vector method of your metric function.

truth  The realized vector of truth. This is either a factor or a numeric.

estimate  The realized estimate result. This is either a numeric vector, a factor vector, or a numeric matrix (in the case of multiple class probability columns) depending on your metric function.

na_rm  A logical value indicating whether NA values should be stripped before the computation proceeds. NA values are removed before getting to your core implementation function so you do not have to worry about handling them yourself. If na_rm=FALSE and any NA values exist, then NA is automatically returned.

cls  A character vector of length 1 or 2 corresponding to the class that truth and estimate should be, respectively. If truth and estimate are of the same class, just supply a vector of length 1. If they are different, supply a vector of length 2. For matrices, it is best to supply "numeric" as the class to check here.

estimator  The type of averaging to use. By this point, the averaging type should be finalized, so this should be a character vector of length 1. By default, this character value is required to be one of: "binary", "macro", "micro", or "macro_weighted". If your metric allows more or less averaging methods, override this with averaging_override.

case_weights  Optionally, the realized case weights, as a numeric vector. This must be the same length as truth, and will be considered in the na_rm checks. If supplied, this will be passed on to metric_impl as the named argument case_weights.

...  Extra arguments to your core metric function, metric_impl, can technically be passed here, but generally the extra args are added through R's scoping rules because the core metric function is created on the fly when the vector method is called.

Details

metric_vec_template() is called from the vector implementation of your metric. Also defined inside your vector implementation is a separate function performing the core implementation of the metric function. This core function is passed along to metric_vec_template() as metric_impl.
**mn_log_loss**

**Mean log loss for multinomial data**

### Description

Compute the logarithmic loss of a classification model.

### Usage

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

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

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. Otherwise, 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.

### See Also

- `metric_summarizer()`  
- `finalize_estimator()`  
- `dots_to_estimate()`
**mn_log_loss**

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

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()`, `classification_cost()`, `gain_capture()`, `pr_auc()`, `roc_auc()`, `roc_aunp()`, `roc_aunu()`
Examples

```r
# 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().
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()`).
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
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

```r
# 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)

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"

npv
Negative predictive value

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(), and ppv().

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></td>
<td>Positive</td>
</tr>
<tr>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>Negative</td>
</tr>
<tr>
<td></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 = \( \frac{Sensitivity \times Prevalence}{(Sensitivity \times Prevalence) + ((1 - Specificity) \times (1 - Prevalence))} \)
- 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(), 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"
)

---

**pathology**

*Liver Pathology Data*

Description

Liver Pathology Data

Details

These data have the results of a 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
  data(pathology)
  str(pathology)

---

poisson_log_loss Mean log loss for Poisson data

Description
  Calculate the loss function for the Poisson distribution.

Usage
  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**

```r
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)
```

---

**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, ...)
```

```r
## 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(),
    ...)
)

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.

... 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 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></th>
<th>Reference</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>Positive</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

The formulas used here are:

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

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

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

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

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

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(),
  ...
)
```

```
precision_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.
precision

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

The formulas used here are:

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

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

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

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, ...)
```

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

```r
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  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 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()`, `classification_cost()`, `gain_capture()`, `mn_log_loss()`, `roc_auc()`, `roc_aunp()`, `roc_aunu()`

Examples

```r
# Two class example

data(two_class_example)

pr_auc(two_class_example, truth, Class1)
```

```r
# Multiclass example

data(hpc_cv)

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.
pr_curve

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_auc_vec(
  truth = fold$obs,
  matrix(  
c(fold$VF, fold$F, fold$M, fold$L),
  ncol = 4
  )
)

pr_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. Otherwise, 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.
- **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.
See Also

Compute the area under the precision recall curve with `pr_auc()`. Other curve metrics: `gain_curve()`, `lift_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.
pr_curve(two_class_example, truth, Class1)

# Visualize the curve using ggplot2 manually
library(ggplot2)
library(dplyr)
pr_curve(two_class_example, truth, Class1) %>%
ggplot(aes(x = recall, y = precision)) +
  geom_path() +
  coord_equal() +
  theme_bw()

# Or use autoplot
autoplot(pr_curve(two_class_example, truth, Class1))

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

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

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.

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 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.

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.

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).

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.

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:

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

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

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

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(), 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"
)

---

## Description

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

## Usage

\texttt{rmse(data, ...)}

\texttt{## S3 method for class 'data.frame'}
\texttt{rmse(data, truth, estimate, na.rm = TRUE, case_weights = NULL, ...)}

\texttt{rmse_vec(truth, estimate, na.rm = TRUE, case_weights = NULL, ...)}

## Arguments

- \texttt{data} A \texttt{data.frame} containing the columns specified by the \texttt{truth} and \texttt{estimate} arguments.
- \texttt{...} Not currently used.
- \texttt{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 \texttt{.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) %>%
    # Supply truth and predictions as bare column names
    rmse(solubility_test, solubility, prediction)
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. Otherwise, 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.

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()`, `classification_cost()`, `gain_capture()`, `mn_log_loss()`, `pr_auc()`, `roc_aunp()`, `roc_aunu()`

Examples

```
# 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. 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, M, VF:L)

# 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() 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_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()`, `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.
da(hpc_cv)

# You can use the col1:colN tidyselect syntax
touch(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_aunp(obs, M, VF:L)

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

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

roc_aunp_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())

truth,
roc_aunu

```r
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

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()`, `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**  
*Receiver operator curve*

**Description**

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. Otherwise, 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.
roc_curve

**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.

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

# 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.

... 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 inter-quartile metric: \texttt{rpiq()}

Other numeric metrics: \texttt{ccc()}, \texttt{huber_loss_pseudo()}, \texttt{huber_loss()}, \texttt{iic()}, \texttt{mae()}, \texttt{mape()},\texttt{mase()}, \texttt{mpe()}, \texttt{msd()}, \texttt{poisson_log_loss()}, \texttt{rmse()}, \texttt{rpiq()}, \texttt{rsq_trad()}, \texttt{rsq()}, \texttt{smape()}

Other consistency metrics: \texttt{ccc()}, \texttt{rpiq()}, \texttt{rsq_trad()}, \texttt{rsq()}

Examples

```r
# 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))
```
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

```r
rpiq(data, ...)  
## 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: `rp()`
Other numeric metrics: `ccc()`, `huber_loss_pseudo()`, `huber_loss()`, `iic()`, `mae()`, `mape()`, `mase()`, `mpe()`, `msd()`, `poisson_log_loss()`, `rmse()`, `rp()`, `rsq_trad()`, `rsq()`, `smape()`
Other consistency metrics: `ccc()`, `rp()`, `rsq_trad()`, `rsq()`

Examples

# Supply truth and predictions as bare column names
rp(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)
m

# 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**

```r
r
rsq(data, ...)
```

```r
## 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 \( \text{.metric} \), \( \text{.estimator} \), and \( \text{.estimate} \) 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
\( \text{solubility\_resampled \leftarrow bind\_rows(} \)
\( \text{replicate(} \)
\( \text{n = times,} \)
\( \text{expr = sample\_n(solubility\_test, size, replace = TRUE),} \)
\( \text{simplify = FALSE} \)
# Compute the metric by group
metric_results <- solubility_resampled %>%
  group_by(resample) %>%
  rsq(solubility, prediction)

metric_results

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

# With uninformitive 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)

# A constant `truth` or `estimate` vector results in a warning from
# a divide by zero error in the correlation calculation.
# `NA` will be returned in these cases.
truth <- c(1, 2)
estimate <- c(1, 1)
rsq_vec(truth, estimate)

---

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**

rsq_trad(data, ...)

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

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()`, `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 uninformitive data, the traditional version of R^2 can return
# negative values.
set.seed(2291)
solubility_test$randomized <- sample(solubility_test$prediction)
srq(solubility_test, solubility, randomized)
srq_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
sens

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
# 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></th>
<th>Reference</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted</td>
<td>Positive</td>
<td>Negative</td>
<td></td>
</tr>
<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:

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

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

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

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

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

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.
- **...**: 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

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

# 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
```r
data(solubility_test)
str(solubility_test)
```

---

**spec**  

**Description**

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,
```

---

**Specificity**
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 \( \# \text{ true_negative} = 0 \) and \( \# \text{ false_positive} = 0 \) 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 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></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:

\[
\text{Sensitivity} = A/(A + C) \\
\text{Specificity} = D/(B + D) \\
\text{Prevalence} = (A + C)/(A + B + C + D) \\
\text{PPV} = (\text{Sensitivity} \times \text{Prevalence})/((\text{Sensitivity} \times \text{Prevalence}) + ((1-\text{Specificity})\times(1-\text{Prevalence}))) \\
\text{NPV} = (\text{Specificity} \times (1-\text{Prevalence}))/(((1-\text{Sensitivity})\times\text{Prevalence}) + ((\text{Specificity} \times (1-\text{Prevalence}))))
\]
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**

```r
# 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 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

```r
## 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)
)

Description

Two Class Predictions

Details

These data are a test set from 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.
two_class_example

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)
Index

* accuracy metrics
  - ccc, 10
  - huber_loss, 34
  - huber_loss_pseudo, 36
  - iic, 38
  - mae, 49
  - mape, 50
  - mase, 52
  - mpe, 67
  - msd, 69
  - poisson_log_loss, 76
  - rmse, 93
  - smape, 119

* class metrics
  - accuracy, 3
  - bal_accuracy, 8
  - detection_prevalence, 18
  - f_meas, 21
  - j_index, 40
  - kap, 43
  - mcc, 55
  - npv, 72
  - ppv, 77
  - precision, 81
  - recall, 90
  - sens, 115
  - spec, 122

* class probability metrics
  - average_precision, 4
  - classification_cost, 13
  - gain_capture, 24
  - mn_log_loss, 65
  - pr_auc, 84
  - roc_auc, 95
  - roc_aunp, 99
  - roc_aunu, 101

* consistency metrics
  - ccc, 10
  - rpd, 106
  - rpiq, 109
  - rsq, 111
  - rsq_trad, 113

* curve metrics
  - gain_curve, 28
  - lift_curve, 46
  - pr_curve, 87
  - roc_curve, 104

* datasets
  - hpc_cv, 33
  - pathology, 75
  - solubility_test, 121
  - two_class_example, 127

* numeric metrics
  - ccc, 10
  - huber_loss, 34
  - huber_loss_pseudo, 36
  - iic, 38
  - mae, 49
  - mape, 50
  - mase, 52
  - mpe, 67
  - msd, 69
  - poisson_log_loss, 76
  - rmse, 93
  - rpd, 106
  - rpiq, 109
  - rsq, 111
  - rsq_trad, 113
  - smape, 119

* relevance metrics
  - f_meas, 21
  - precision, 81
  - recall, 90

* sensitivity metrics
  - npv, 72
  - ppv, 77
  - sens, 115
  - spec, 122
accuracy, 3, 10, 20, 23, 42, 45, 56, 74, 80, 83, 92, 118, 125
accuracy(), 43, 57, 66, 126
accuracy_vec(accuracy), 3
average_precision, 4, 15, 26, 66, 86, 97, 100, 103
average_precision_vec
(average_precision), 4
bal_accuracy, 4, 8, 20, 23, 42, 45, 56, 74, 80, 83, 92, 118, 125
bal_accuracy_vec(bal_accuracy), 8
base::table(), 16
ccc(), 11
ccc_vec(ccc), 10
classification_cost, 7, 13, 26, 66, 86, 97, 100, 103
classification_cost_vec
(classification_cost), 13
classification_cost_vec
(classification_cost), 13
conf_mat, 16
conf_mat(), 17, 126, 127
detection_prevalence, 4, 10, 18, 23, 42, 45, 56, 74, 80, 83, 92, 118, 125
detection_prevalence_vec
(detection_prevalence), 18
developer-helpers(get_weights), 31
dots_to_estimate(get_weights), 31
dots_to_estimate(), 61, 62, 65
f_meas, 4, 10, 20, 21, 42, 45, 56, 74, 80, 83, 92, 118, 125
f_meas(), 21, 62, 81, 90, 126
f_meas_vec(f_meas), 21
finalize_estimator(get_weights), 31
finalize_estimator(), 62, 65
finalize_estimator_internal
(get_weights), 31
gain_capture, 7, 15, 24, 66, 86, 97, 100, 103
gain_capture(), 28, 30
gain_capture_vec(gain_capture), 24
gain_curve, 28, 48, 89, 105
gain_curve(), 26, 46
get_weights, 31
ggplot2::autoplot(), 17, 29, 47, 88, 105
hpc_cv, 33
huber_loss(), 36
huber_loss_pseudo_vec
(huber_loss_pseudo), 36
huber_loss_vec(huber_loss), 34
iic_vec(iic), 38
j_index, 4, 10, 20, 23, 40, 45, 56, 74, 80, 83, 92, 118, 125
j_index_vec(j_index), 40
kap, 4, 10, 20, 23, 42, 43, 56, 74, 80, 83, 92, 118, 125
kap(), 57
kap_vec(kap), 43
lift_curve, 30, 46, 89, 105
lift_curve(), 28
mae(), 58, 69, 70
mae_vec(mae), 49
mape_vec(mape), 50
mase_vec(mase), 52
mcc, 4, 10, 20, 23, 42, 45, 55, 74, 80, 83, 92, 118, 125
mcc_vec(mcc), 55
metric_set, 58
metric_set(), 58, 62, 71
metric_summarizer, 61
metric_summarizer(), 31, 33, 63, 65
metric_tweak, 62
metric_vec_template, 63
metric_vec_template(), 31, 33, 61, 62
metrics, 57
metrics(), 59
mn_log_loss, 7, 15, 26, 65, 86, 97, 100, 103
mn_log_loss(), 57, 58
mn_log_loss_vec (mn_log_loss), 65
mpe, 12, 35, 37, 39, 50, 51, 54, 67, 71, 77, 94,
108, 110, 112, 114, 120
mpe_vec (mpe), 67
msd, 12, 35, 37, 39, 50, 51, 54, 68, 69, 77, 94,
108, 110, 112, 114, 120
msd_vec (msd), 69

new-metric, 71
new_class_metric (new-metric), 71
new_numeric_metric (new-metric), 71
new_prob_metric (new-metric), 71

ppv, 4, 10, 20, 23, 42, 45, 56, 72, 80, 83, 92,
118, 125
ppv(), 72, 73, 77, 79, 115, 122
ppv_vec (ppv), 72

pathology, 75

poisson_log_loss, 12, 35, 37, 39, 50, 51, 54,
68, 71, 76, 94, 108, 110, 112, 114, 120

poisson_log_loss_vec (poisson_log_loss), 76

ppv, 4, 10, 20, 23, 42, 45, 56, 74, 77, 83, 92,
118, 125
ppv(), 72, 73, 77, 79, 115, 122
ppv_vec (ppv), 77

pr_auc, 7, 15, 26, 66, 84, 97, 100, 103
pr_auc(), 7, 87, 89
pr_auc_vec (pr_auc), 84

pr_auc_vec (pr_auc), 84
pr_curve, 30, 48, 87, 105
pr_curve(), 4, 6, 7, 84, 86

precision, 4, 10, 20, 23, 42, 45, 56, 74, 80,
81, 92, 118, 125
precision(), 21, 81, 90
precision_vec (precision), 81

quasiquote, 3, 5, 9, 11, 14, 16, 19, 22,
25, 29, 35, 37, 39, 41, 44, 46, 49, 51,
53, 55, 57, 65, 68, 70, 73, 76, 78, 81,
85, 88, 90, 93, 96, 99, 102, 104, 107,
109, 111, 114, 117, 120, 123

recall, 4, 10, 20, 23, 42, 45, 56, 74, 80, 83,
90, 118, 125
recall(), 21, 81, 90, 126
recall_vec (recall), 90

rmse, 12, 35, 37, 39, 50, 51, 54, 68, 71, 77, 93,
108, 110, 112, 114, 120
rmse(), 11, 34, 36, 38
rmse_vec (rmse), 93
roc_auc, 7, 15, 26, 66, 86, 95, 100, 103
roc_auc(), 57, 58, 104, 105
roc_auc_vec (roc_auc), 95
roc_aunp, 7, 15, 26, 66, 86, 97, 99, 103
roc_aunp(), 103
roc_aunp_vec (roc_aunp), 99
roc_aucn, 7, 15, 26, 66, 86, 97, 100, 101
roc_aucn(), 100
roc_aucn_vec (roc_aucn), 101
roc_curve, 30, 48, 89, 104
roc_curve(), 95, 97

rdp, 12, 35, 37, 39, 50, 51, 54, 68, 71, 77, 94,
106, 110, 112, 114, 120
rdp(), 107, 109, 110
rdp_vec (rdp), 106

rpiq, 12, 35, 37, 39, 50, 51, 54, 68, 71, 77, 94,
108, 109, 112, 114, 120
rpiq(), 107–109
rpiq_vec (rpiq), 109

rsq, 12, 35, 37, 39, 50, 51, 54, 68, 71, 77, 94,
108, 110, 111, 114, 120
rsq(), 11, 58, 112–114

rsq_trad, 12, 35, 37, 39, 50, 51, 54, 68, 71,
77, 94, 108, 110, 112, 113, 120
rsq_trad(), 111, 112, 114
rsq_trad_vec (rsq_trad), 113
rsq_vec (rsq), 111

sens, 4, 10, 20, 23, 42, 45, 56, 74, 80, 83, 92,
115, 125
sens(), 8, 40, 72, 77, 115, 122, 126
sens_vec (sens), 115
sensitivity (sens), 115
sensitivity_vec (sens), 115

smape, 12, 35, 37, 39, 50, 51, 54, 68, 71, 77,
smape_vec (smape), 119

solubility_test, 121
spec, 4, 10, 20, 23, 42, 45, 56, 74, 80, 83, 92,
118, 122
spec(), 8, 40, 72, 77, 115, 122
spec_vec (spec), 122
specificity (spec), 122
specificity_vec (spec), 122
summary.conf.mat, 126
summary.conf_mat(), 17

tidy.conf_mat(conf_mat), 16
two_class_example, 127

validate_estimator(get_weights), 31