Package ‘yardstick’

March 8, 2019

Type Package

Title Tidy Characterizations of Model Performance

Version 0.0.3

Maintainer Max Kuhn <max@rstudio.com>

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).

URL https://github.com/tidymodels/yardstick

BugReports https://github.com/tidymodels/yardstick/issues

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Depends R (>= 2.10)

Suggests testthat, covr, purrr, tidyr, ggplot2, knitr, kableExtra, rmarkdown

Imports dplyr, utils, pROC, rlang, tidyselect, Rcpp, generics

LinkingTo Rcpp

VignetteBuilder knitr

NeedsCompilation yes

Author Max Kuhn [aut, cre],
Davis Vaughan [aut],
RStudio [cph]

Repository CRAN

Date/Publication 2019-03-08 19:00:03 UTC

R topics documented:

accuracy *Accuracy*

Description

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

Usage

```
accuracy(data, ...)

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

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

Arguments

`data`	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

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

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

# Multiclass
library(dplyr)
data(hpc_cv)

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

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

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

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

# Making Class2 the "relevant" level
options(yardstick.event_first = FALSE)
accuracy_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)
```

---

bal_accuracy *Balanced accuracy*

Description

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

Usage

```
bal_accuracy(data, ...)

## S3 method for class 'data.frame'
```

```
bal_accuracy(data, truth, estimate,
  estimator = NULL, na_rm = TRUE, ...)

bal_accuracy_vec(truth, estimate, estimator = NULL, na_rm = TRUE, ...)
```

Arguments

data	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

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

Author(s)

Max Kuhn

See Also

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

Examples

```
# 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
options(yardstick.event_first = FALSE)
bal_accuracy_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)
```

`ccc` *Concordance correlation coefficient*

Description

Calculate the concordance correlation coefficient.

Usage

```
ccc(data, ...)

## S3 method for class 'data.frame'
```

```
ccc(data, truth, estimate, bias = FALSE,
  na_rm = TRUE, ...)

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

Arguments

data	A data.frame containing the truth and estimate columns.
...	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.

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

Lin, L. (1989). A concordance correlation coefficient to evaluate reproducibility. *Biometrics*, 45 (1), 255-268.

Nickerson, C. (1997). A note on "A concordance correlation coefficient to evaluate reproducibility". *Biometrics*, 53(4), 1503-1507.

See Also

Other numeric metrics: huber_loss_pseudo, huber_loss, mae, mape, mase, rmse, rpd, rpiq, rsq_trad, rsq, smape

Other consistency metrics: rpd, rpiq, rsq_trad, rsq

Other accuracy metrics: huber_loss_pseudo, huber_loss, mae, mape, mase, rmse, smape

Examples

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

library(dplyr)

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

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

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

metric_results

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

---

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"), ...)

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

```
## S3 method for class 'conf_mat'
autoplot(object, type = "mosaic", ...)
```

Arguments

data	A data frame or a `base::table()`.
...	Options to pass to `base::table()` (not including dnn). This argument is not currently used for the tidy method.
truth	The column identifier for the true class results (that is a `factor`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `factor` vector.
estimate	The column identifier for the predicted class results (that is also `factor`). As with `truth` this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a `factor` vector.
dnn	A character vector of dimnames for the table.
x	A `conf_mat` object.
object	The `conf_mat` data frame returned from `conf_mat()`.
type	Type of plot desired, must be "mosaic" or "heatmap", defaults to "mosaic".

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)
cm

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

cells_per_resample <- hpc_cv %>%
  group_by(Resample) %>%
  conf_mat(obs, pred) %>%
  mutate(tidied = map(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, ...)

detection_prevalence_vec(truth, estimate, estimator = NULL,
  na_rm = TRUE, ...)
```

Arguments

`data`	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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)

# Making Class2 the "relevant" level
options(yardstick.event_first = FALSE)
detection_prevalence_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)
```

`f_meas` *F Measure*

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, ...)

f_meas_vec(truth, estimate, beta = 1, estimator = NULL, na_rm = TRUE,
  ...)
```

Arguments

`data`	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

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. In yardstick, the default is to use the *first* level. To change this, a global option called yardstick.event_first is set to TRUE when the package is loaded. This can be changed to FALSE if the last level of the factor is considered 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:

	Reference	
Predicted	Relevant	Irrelevant
Relevant	A	B
Irrelevant	C	D

The formulas used here are:

$$recall = A/(A + C)$$

$$precision = A/(A + B)$$

$$F_meas_\beta = (1 + \beta^2) * precision * recall/((\beta^2 * precision) + recall)$$

See the references for discussions of the statistics.

Author(s)

Max Kuhn

References

Buckland, M., & Gey, F. (1994). The relationship between Recall and Precision. *Journal of the American Society for Information Science*, 45(1), 12-19.

Powers, D. (2007). Evaluation: From Precision, Recall and F Factor to ROC, Informedness, Markedness and Correlation. Technical Report SIE-07-001, Flinders University

See Also

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

Other relevance metrics: precision, recall

Examples

```
# 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
options(yardstick.event_first = FALSE)
f_meas_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)
```

---

gain_capture *Gain capture*

Description

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

Usage

```
gain_capture(data, ...)

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

gain_capture_vec(truth, estimate, estimator = NULL, na_rm = TRUE, ...)
```

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.
`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

`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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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

Engelmann, Bernd & Hayden, Evelyn & Tasche, Dirk (2003). "Measuring the Discriminative Power of Rating Systems," Discussion Paper Series 2: Banking and Financial Studies 2003,01, Deutsche Bundesbank.

See Also

`gain_curve()` to compute the full gain curve.

Other class probability metrics: `mn_log_loss`, `pr_auc`, `roc_auc`

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

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

## S3 method for class 'gain_df'
autoplot(object, ...)
```

Arguments

data	A data.frame containing the truth and estimate columns.
...	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.

`object` — The `gain_df` data frame returned from `gain_curve()`.

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.

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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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

Engelmann, Bernd & Hayden, Evelyn & Tasche, Dirk (2003). "Measuring the Discriminative Power of Rating Systems," Discussion Paper Series 2: Banking and Financial Studies 2003,01, Deutsche Bundesbank.

See Also

Compute the relevant area under the gain curve with `gain_capture()`.

Other curve metrics: `lift_curve`, `pr_curve`, `roc_curve`

Examples

```
library(ggplot2)
library(dplyr)

# Two class - a tibble is returned
gain_curve(two_class_example, truth, Class1)

# 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 `vignette("custom-metrics", "yardstick")` for more information.

Usage

```
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", "ma` Set this is 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:
   - factor - Then "binary" is returned if it has 2 levels, otherwise "macro" is returned.
   - numeric - Then "binary" is returned.
   - table - Then "binary" is returned if it has 2 columns, otherwise "macro" is returned. This is useful if you have table methods.
   - matrix - Then "macro" is returned.

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()

hpc_cv *Class Probability Predictions*

Description

Class Probability Predictions

Details

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

Value

hpc_cv	a data frame

Source

Kuhn, M., Johnson, K. (2013) *Applied Predictive Modeling*, Springer

Examples

```
data(hpc_cv)
str(hpc_cv)
```

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

```
huber_loss(data, ...)

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

huber_loss_vec(truth, estimate, delta = 1, na_rm = TRUE, ...)
```

Arguments

data	A data.frame containing the truth and estimate columns.
...	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.

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

Huber, P. (1964). Robust Estimation of a Location Parameter. *Annals of Statistics*, 53 (1), 73-101.

See Also

Other numeric metrics: ccc, huber_loss_pseudo, mae, mape, mase, rmse, rpd, rpiq, rsq_trad, rsq, smape

Other accuracy metrics: ccc, huber_loss_pseudo, mae, mape, mase, 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, ...)

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

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.

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

Huber, P. (1964). Robust Estimation of a Location Parameter. *Annals of Statistics*, 53 (1), 73-101.

Hartley, Richard (2004). Multiple View Geometry in Computer Vision. (Second Edition). Page 619.

See Also

Other numeric metrics: ccc, huber_loss, mae, mape, mase, rmse, rpd, rpiq, rsq_trad, rsq, smape

Other accuracy metrics: ccc, huber_loss, mae, mape, mase, 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))
```

---

j_index *J-index*

Description

Youden's J statistic is defined as:

sens() + 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, estimate, estimator = NULL,
  na_rm = TRUE, ...)

j_index_vec(truth, estimate, estimator = NULL, na_rm = TRUE, ...)
```

Arguments

data	Either a data.frame containing the truth and estimate columns, 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.

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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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

Youden, W.J. (1950). "Index for rating diagnostic tests". Cancer. 3: 32-35.

Powers, David M W (2011). "Evaluation: From Precision, Recall and F-Score to ROC, Informedness, Markedness and Correlation". Journal of Machine Learning Technologies. 2 (1): 37-63.

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
options(yardstick.event_first = FALSE)
j_index_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)
```

---

kap *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, na_rm = TRUE, ...)

kap_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

data	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

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

References

Cohen, J. (1960). "A coefficient of agreement for nominal scales". *Educational and Psychological Measurement*. 20 (1): 37-46.

See Also

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

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)
```

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

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

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

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

# Making Class2 the "relevant" level
options(yardstick.event_first = FALSE)
kap_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)
```

`lift_curve` *Lift curve*

Description

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

Usage

```
lift_curve(data, ...)

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

## S3 method for class 'lift_df'
autoplot(object, ...)
```

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.
object	The `lift_df` data frame returned from `lift_curve()`.

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`.

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 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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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

```
library(ggplot2)
library(dplyr)

# Two class - a tibble is returned
lift_curve(two_class_example, truth, Class1)

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

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

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

---

mae *Mean absolute error*

Description

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

Usage

```
mae(data, ...)

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

mae_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

data	A data.frame containing the truth and estimate columns.
...	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.

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, mape, mase, rmse, rpd, rpiq, rsq_trad, rsq, smape

Other accuracy metrics: ccc, huber_loss_pseudo, huber_loss, mape, mase, rmse, smape

Examples

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

library(dplyr)

set.seed(1234)
size <- 100
```

```
times <- 10

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

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

metric_results

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

---

mape *Mean absolute percent error*

Description

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

Usage

```
mape(data, ...)

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

mape_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.

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`, `mae`, `mase`, `rmse`, `rpd`, `rpiq`, `rsq_trad`, `rsq`, `smape`

Other accuracy metrics: `ccc`, `huber_loss_pseudo`, `huber_loss`, `mae`, `mase`, `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 neccesary to order observations in ascending order by time.

Usage

```
mase(data, ...)

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

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

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.

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

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

See Also

Other numeric metrics: ccc, huber_loss_pseudo, huber_loss, mae, mape, rmse, rpd, rpiq, rsq_trad, rsq, smape

Other accuracy metrics: ccc, huber_loss_pseudo, huber_loss, mae, mape, 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, ...)

mcc_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

`data`	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

Value

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

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

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

Relevant level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result. In yardstick, the default is to use the *first* level. To change this, a global option called yardstick.event_first is set to TRUE when the package is loaded. This can be changed to FALSE if the last level of the factor is considered the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

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

Author(s)

Max Kuhn

References

Giuseppe, J. (2012). "A Comparison of MCC and CEN Error Measures in Multi-Class Prediction". *PLOS ONE*. Vol 7, Iss 8, e41882.

See Also

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

Examples

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

# Multiclass
# mcc() has a natural multiclass extension
library(dplyr)
data(hpc_cv)
hpc_cv %>%
  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

```
metrics(data, ...)

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

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns and any columns specified by `...`.
`...`	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.
`options`	A `list` of named options to pass to `pROC::roc()` such as `direction` or `smooth`. These options should not include `response`, `predictor`, or `levels`.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.

Value

A two column tibble.

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

See Also

metric_set()

Examples

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

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

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

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

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

---

metric_set — *Combine metric functions*

Description

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

Usage

```
metric_set(...)
```

Arguments

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

Details

All functions must be either:

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

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

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

Numeric metrics will have a signature like: `fn(data, truth, estimate, na_rm = TRUE, ...)`.

Class/prob metrics have a signature of `fn(data, truth, ..., estimate, na_rm = TRUE)`. 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

```
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.
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,
    ...
  )
}
```

```
# Add on the underlying function class (here, "numeric_metric")
class(ccc_with_bias) <- class(ccc)

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 `vignette("custom-metrics", "yardstick")` for more information.

Usage

```
metric_summarizer(metric_nm, metric_fn, data, truth, estimate,
  estimator = NULL, na_rm = TRUE, ..., 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 NA 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().
...	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()

---

metric_vec_template *Developer function for calling new metrics*

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, ...)
```

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`.
`...`	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`.

See Also

`metric_summarizer()` `finalize_estimator()` `dots_to_estimate()`

`mn_log_loss` *Mean log loss*

Description

Compute the logarithmic loss of a classification model.

Usage

```
mn_log_loss(data, ...)

## S3 method for class 'data.frame'
mn_log_loss(data, truth, ..., na_rm = TRUE,
  sum = FALSE)

mn_log_loss_vec(truth, estimate, na_rm = TRUE, sum = FALSE, ...)
```

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.
`sum`	A `logical`. Should the sum of the likelihood contributions be returned (instead of the mean value)?
`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: gain_capture, pr_auc, roc_auc

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

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

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


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

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

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

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, ...)

npv_vec(truth, estimate, prevalence = NULL, estimator = NULL,
  na_rm = TRUE, ...)
```

Arguments

data	Either a data.frame containing the truth and estimate columns, 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.

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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

$$Sensitivity = A/(A + C)$$

$$Specificity = D/(B + D)$$

$$Prevalence = (A + C)/(A + B + C + D)$$

$$PPV = (Sensitivity * Prevalence)/((Sensitivity * Prevalence) + ((1 - Specificity) * (1 - Prevalence)))$$

$$NPV = (Specificity * (1 - Prevalence))/(((1 - Sensitivity) * Prevalence) + ((Specificity) * (1 - Prevalence)))$$

See the references for discussions of the statistics.

Author(s)

Max Kuhn

References

Altman, D.G., Bland, J.M. (1994) "Diagnostic tests 2: predictive values," *British Medical Journal*, vol 309, 102.

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
options(yardstick.event_first = FALSE)
npv_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)
```

`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

Altman, D.G., Bland, J.M. (1994) “Diagnostic tests 1: sensitivity and specificity,” *British Medical Journal*, vol 308, 1552.

Examples

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

---

ppv *Positive predictive value*

Description

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

Usage

```
ppv(data, ...)

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

ppv_vec(truth, estimate, prevalence = NULL, estimator = NULL,
  na_rm = TRUE, ...)
```

Arguments

`data`	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

$$Sensitivity = A/(A + C)$$

$$Specificity = D/(B + D)$$

$$Prevalence = (A + C)/(A + B + C + D)$$

$$PPV = (Sensitivity * Prevalence)/((Sensitivity * Prevalence) + ((1 - Specificity) * (1 - Prevalence)))$$

$$NPV = (Specificity * (1 - Prevalence))/(((1 - Sensitivity) * Prevalence) + ((Specificity) * (1 - Prevalence)))$$

See the references for discussions of the statistics.

Author(s)

Max Kuhn

References

Altman, D.G., Bland, J.M. (1994) "Diagnostic tests 2: predictive values," *British Medical Journal*, vol 309, 102.

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
options(yardstick.event_first = FALSE)
ppv_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)

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

---

precision *Precision*

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, ...)

precision_vec(truth, estimate, estimator = NULL, na_rm = TRUE, ...)
```

Arguments

`data`	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

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).

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. In yardstick, the default is to use the *first* level. To change this, a global option called yardstick.event_first is set to TRUE when the package is loaded. This can be changed to FALSE if the last level of the factor is considered 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:

	Reference	
Predicted	Relevant	Irrelevant
Relevant	A	B
Irrelevant	C	D

The formulas used here are:

$$recall = A/(A + C)$$

$$precision = A/(A + B)$$

$$F_meas_\beta = (1 + \beta^2) * precision * recall/((\beta^2 * precision) + recall)$$

See the references for discussions of the statistics.

Author(s)

Max Kuhn

References

Buckland, M., & Gey, F. (1994). The relationship between Recall and Precision. *Journal of the American Society for Information Science*, 45(1), 12-19.

Powers, D. (2007). Evaluation: From Precision, Recall and F Factor to ROC, Informedness, Markedness and Correlation. Technical Report SIE-07-001, Flinders University

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

```
# 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
options(yardstick.event_first = FALSE)
precision_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)
```

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

```
pr_auc(data, ...)

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

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

Arguments

data	A data.frame containing the truth and estimate columns.
...	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.
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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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: `gain_capture`, `mn_log_loss`, `roc_auc`

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

# You can use the col1:colN tidyselect syntax
hpc_cv %>%
  filter(Resample == "Fold01") %>%
  pr_auc(obs, 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 = fold1$obs,
   matrix(
     c(fold1$VF, fold1$F, fold1$M, fold1$L),
     ncol = 4
   )
)
```

`pr_curve` *Precision recall curve*

Description

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

Usage

```
pr_curve(data, ...)

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

## S3 method for class 'pr_df'
autoplot(object, ...)
```

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.
`object`	The `pr_df` data frame returned from `pr_curve()`.

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. In yardstick, the default is to use the *first* level. To change this, a global option called yardstick.event_first is set to TRUE when the package is loaded. This can be changed to FALSE if the last level of the factor is considered the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Author(s)

Max Kuhn

See Also

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

Other curve metrics: gain_curve, lift_curve, roc_curve

Examples

```
library(ggplot2)
library(dplyr)

# Two class - a tibble is returned
pr_curve(two_class_example, truth, Class1)

# Visualize the curve using ggplot2 manually
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()
```

---

recall *Recall*

Description

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

Usage

```
recall(data, ...)

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

recall_vec(truth, estimate, estimator = NULL, na_rm = TRUE, ...)
```

Arguments

`data`	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

Details

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

Value

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

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

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

Relevant level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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:

	Reference	
Predicted	Relevant	Irrelevant
Relevant	A	B
Irrelevant	C	D

The formulas used here are:

$$recall = A/(A + C)$$

$$precision = A/(A + B)$$

$$F_meas_\beta = (1 + \beta^2) * precision * recall/((\beta^2 * precision) + recall)$$

See the references for discussions of the statistics.

Author(s)

Max Kuhn

References

Buckland, M., & Gey, F. (1994). The relationship between Recall and Precision. *Journal of the American Society for Information Science*, 45(1), 12-19.

Powers, D. (2007). Evaluation: From Precision, Recall and F Factor to ROC, Informedness, Markedness and Correlation. Technical Report SIE-07-001, Flinders University

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

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

# 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
options(yardstick.event_first = FALSE)
recall_vec(two_class_example$truth, two_class_example$predicted)
```

```
options(yardstick.event_first = TRUE)
```

---

rmse *Root mean squared error*

Description

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

Usage

```
rmse(data, ...)

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

rmse_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.

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, mae, mape, mase, rpd, rpiq, rsq_trad, rsq, smape

Other accuracy metrics: ccc, huber_loss_pseudo, huber_loss, mae, mape, mase, smape

Examples

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

library(dplyr)

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

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

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

metric_results

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

---

roc_auc *Area under the receiver operator curve*

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, ..., options = list(),
  estimator = NULL, na_rm = TRUE)

roc_auc_vec(truth, estimate, options = list(), estimator = NULL,
  na_rm = TRUE, ...)
```

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.
`options`	A `list` of named options to pass to `pROC::roc()` such as `direction` or `smooth`. These options should not include `response`, `predictor`, or `levels`.
`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"` or `"hand_till"` based on `truth`.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`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

For most methods, `roc_auc()` makes no effort to ensure that the supplied class probabilities result in a AUC value above `0.5` (random guessing). However, the Hand, Till (2001) method assumes that the individual AUCs are all above `0.5`, so if an AUC value below `0.5` is computed, then `1` is subtracted from it to get the correct result.

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`.

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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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.

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

Hand, Till (2001). "A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems". *Machine Learning*. Vol 45, Iss 2, pp 171-186.

Fawcett (2005). "An introduction to ROC analysis". *Pattern Recognition Letters*. 27 (2006), pp 861-874.

Provost, F., Domingos, P., 2001. "Well-trained PETs: Improving probability estimation trees", CeDER Working Paper #IS-00-04, Stern School of Business, New York University, NY, NY 10012.

See Also

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

Other class probability metrics: `gain_capture`, `mn_log_loss`, `pr_auc`

Examples

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

# Multiclass
```

```
library(dplyr)
data(hpc_cv)

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

# passing options via a list and _not_ `...`
roc_auc(two_class_example, truth = truth, Class1,
        options = list(smooth = TRUE))
```

`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, ..., options = list(),
```

```
  na_rm = TRUE)

## S3 method for class 'roc_df'
autoplot(object, ...)
```

Arguments

data	A data.frame containing the truth and estimate columns.
...	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.
options	A list of named options to pass to pROC::roc() such as direction or smooth. These options should not include response, predictor, or levels.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
object	The roc_df data frame returned from roc_curve().

Details

roc_curve() computes the sensitivity at every unique value of the probability column (in addition to infinity and minus infinity). If a smooth ROC curve was produced, the unique observed values of the specificity are used to create the curve points. In either case, this may not be efficient for large data sets.

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 specificity and sensitivity.

If an ordinary (i.e. non-smoothed) curve is used, there is also a column for .threshold.

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. In yardstick, the default is to use the *first* level. To change this, a global option called yardstick.event_first is set to TRUE when the package is loaded. This can be changed to FALSE if the last level of the factor is considered 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

```
library(ggplot2)
library(dplyr)

# Two class - a tibbble is returned
roc_curve(two_class_example, truth, Class1)

# Visualize the curve using ggplot2 manually
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, ...)

rpd_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.

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

Williams, P.C. (1987) Variables affecting near-infrared reflectance spectroscopic analysis. In: Near Infrared Technology in the Agriculture and Food Industries. 1st Ed. P.Williams and K.Norris, Eds. Am. Cereal Assoc. Cereal Chem., St. Paul, MN.

Bellon-Maurel, V., Fernandez-Ahumada, E., Palagos, B., Roger, J.M. and McBratney, A., (2010). Critical review of chemometric indicators commonly used for assessing the quality of the prediction of soil attributes by NIR spectroscopy. TrAC Trends in Analytical Chemistry, 29(9), pp.1073-1081.

See Also

The closely related inter-quartile metric: `rpiq()`

Other numeric metrics: `ccc`, `huber_loss_pseudo`, `huber_loss`, `mae`, `mape`, `mase`, `rmse`, `rpiq`, `rsq_trad`, `rsq`, `smape`

Other consistency metrics: `ccc`, `rpiq`, `rsq_trad`, `rsq`

Examples

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

library(dplyr)

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

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

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

metric_results

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

rpiq *Ratio of performance to inter-quartile*

Description

These functions are appropriate for cases where the model outcome is a numeric. The ratio of performance to deviation (rpd()) and the ratio of performance to inter-quartile (rpiq()) are both measures of consistency/correlation between observed and predicted values (and not of accuracy).

Usage

```
rpiq(data, ...)

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

rpiq_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

data	A data.frame containing the truth and estimate columns.
...	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.

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

Williams, P.C. (1987) Variables affecting near-infrared reflectance spectroscopic analysis. In: Near Infrared Technology in the Agriculture and Food Industries. 1st Ed. P.Williams and K.Norris, Eds. Am. Cereal Assoc. Cereal Chem., St. Paul, MN.

Bellon-Maurel, V., Fernandez-Ahumada, E., Palagos, B., Roger, J.M. and McBratney, A., (2010). Critical review of chemometric indicators commonly used for assessing the quality of the prediction of soil attributes by NIR spectroscopy. TrAC Trends in Analytical Chemistry, 29(9), pp.1073-1081.

See Also

The closely related deviation metric: `rpd()`

Other numeric metrics: `ccc`, `huber_loss_pseudo`, `huber_loss`, `mae`, `mape`, `mase`, `rmse`, `rpd`, `rsq_trad`, `rsq`, `smape`

Other consistency metrics: `ccc`, `rpd`, `rsq_trad`, `rsq`

Examples

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

library(dplyr)

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

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

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

metric_results

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

rsq *R squared*

Description

Calculate the coefficient of determination using correlation. For the traditional measure of R squared, see rsq_trad().

Usage

```
rsq(data, ...)

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

rsq_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

data	A data.frame containing the truth and estimate columns.
...	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.

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.

rsq() is simply the squared correlation between truth and estimate.

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_vec(), a single numeric value (or NA).

Author(s)

Max Kuhn

References

Kvalseth. Cautionary note about R^2. American Statistician (1985) vol. 39 (4) pp. 279-285.

See Also

Other numeric metrics: ccc, huber_loss_pseudo, huber_loss, mae, mape, mase, rmse, rpd, rpiq, rsq_trad, smape

Other consistency metrics: ccc, rpd, rpiq, rsq_trad

Examples

```
# Supply truth and predictions as bare column names
rsq(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(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)
```

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, ...)

rsq_trad_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

`data`	A `data.frame` containing the `truth` and `estimate` columns.
`...`	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.

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

Kvalseth. Cautionary note about R^2. American Statistician (1985) vol. 39 (4) pp. 279-285.

See Also

Other numeric metrics: ccc, huber_loss_pseudo, huber_loss, mae, mape, mase, rmse, rpd, rpiq, rsq, smape

Other consistency metrics: ccc, rpd, rpiq, rsq

Examples

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

library(dplyr)

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

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

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

metric_results

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

sens *Sensitivity*

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, ...)

sens_vec(truth, estimate, estimator = NULL, na_rm = TRUE, ...)
```

Arguments

`data`	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

Details

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

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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

$$Sensitivity = A/(A + C)$$

$$Specificity = D/(B + D)$$

$$Prevalence = (A + C)/(A + B + C + D)$$

$$PPV = (Sensitivity * Prevalence)/((Sensitivity * Prevalence) + ((1 - Specificity) * (1 - Prevalence)))$$

$$NPV = (Specificity * (1 - Prevalence))/(((1 - Sensitivity) * Prevalence) + ((Specificity) * (1 - Prevalence)))$$

See the references for discussions of the statistics.

Author(s)

Max Kuhn

References

Altman, D.G., Bland, J.M. (1994) "Diagnostic tests 1: sensitivity and specificity," *British Medical Journal*, vol 308, 1552.

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
options(yardstick.event_first = FALSE)
sens_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)
```

---

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, ...)

smape_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

data	A data.frame containing the truth and estimate columns.
...	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.

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, mae, mape, mase, rmse, rpd, rpiq, rsq_trad, rsq

Other accuracy metrics: ccc, huber_loss_pseudo, huber_loss, mae, mape, mase, rmse

Examples

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

library(dplyr)

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

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

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

metric_results

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

`solubility_test` *Solubility Predictions from MARS Model*

Description

Solubility Predictions from MARS Model

Details

For the solubility data in Kuhn and Johnson (2013), these data are the test set results for the MARS model. The observed solubility (in column `solubility`) and the model results (`prediction`) are contained in the data.

Value

`solubility_test`
: a data frame

Source

Kuhn, M., Johnson, K. (2013) *Applied Predictive Modeling*, Springer

Examples

```
data(solubility_test)
str(solubility_test)
```

`spec` *Specificity*

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

```
spec(data, ...)

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

spec_vec(truth, estimate, estimator = NULL, na_rm = TRUE, ...)
```

Arguments

`data`	Either a `data.frame` containing the `truth` and `estimate` columns, 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.

Details

The specificity measures the proportion of negatives that are correctly identified as negatives. When there are no negative results, specificity is not defined and a value of `NA` is returned.

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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

$$Sensitivity = A/(A + C)$$

$$Specificity = D/(B + D)$$

$$Prevalence = (A + C)/(A + B + C + D)$$

$$PPV = (Sensitivity*Prevalence)/((Sensitivity*Prevalence)+((1-Specificity)*(1-Prevalence)))$$

$$NPV = (Specificity*(1-Prevalence))/(((1-Sensitivity)*Prevalence)+((Specificity)*(1-Prevalence)))$$

See the references for discussions of the statistics.

Author(s)

Max Kuhn

References

Altman, D.G., Bland, J.M. (1994) "Diagnostic tests 1: sensitivity and specificity," *British Medical Journal*, vol 308, 1552.

See Also

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

Other sensitivity metrics: npv, ppv, sens

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

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

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

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

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

# Making Class2 the "relevant" level
options(yardstick.event_first = FALSE)
spec_vec(two_class_example$truth, two_class_example$predicted)
options(yardstick.event_first = TRUE)
```

summary.conf_mat — *Summary Statistics for Confusion Matrices*

Description

Various statistical summaries of confusion matrices are produced and returned in a tibble. These include those shown in the help pages for `sens()`, `recall()`, and `accuracy()`, among others.

Usage

```
## S3 method for class 'conf_mat'
summary(object, prevalence = NULL, beta = 1,
  estimator = NULL, ...)
```

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`.
`...`	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. In `yardstick`, the default is to use the *first* level. To change this, a global option called `yardstick.event_first` is set to `TRUE` when the package is loaded. This can be changed to `FALSE` if the last level of the factor is considered 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(purrr)
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 = map(conf_mat, summary)) %>%
  unnest(summary_tbl)

all_metrics %>%
  group_by(.metric) %>%
  summarise(
    mean = mean(.estimate, na.rm = TRUE),
    sd = sd(.estimate, na.rm = TRUE)
  )
```

`two_class_example` *Two Class Predictions*

Description

Two Class Predictions

Details

These data are a test set form a model built for two classes ("Class1" and "Class2"). There are columns for the true and predicted classes and column for the probabilities for each class.

Value

`two_class_example`
a data frame

Examples

```
data(two_class_example)
str(two_class_example)
```

Index