Package 'yardstick'

April 7, 2026

Type Package

Title Tidy Characterizations of Model Performance

Version 1.4.0

Description Tidy tools for quantifying how well model fits to a data set such as confusion matrices, class probability curve summaries, and regression metrics (e.g., RMSE).

License MIT + file LICENSE

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

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

Depends R (>= 4.1)

Imports cli, dplyr (>= 1.1.0), generics (>= 0.1.2), hardhat (>= 1.4.3), lifecycle (>= 1.0.3), rlang (>= 1.1.4), tibble, tidyselect (>= 1.2.0), utils, vctrs (>= 0.5.0), withr

Suggests covr, ggplot2, knitr, probably (>= 1.0.0), rmarkdown, survival (>= 3.5-0), testthat (>= 3.0.0), tidyr

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-04-24

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

Collate 'aaa-get_metrics.R' 'aaa-metric_set.R' 'aaa-metrics.R' 'import-standalone-types-check.R' 'aaa-new.R' 'aaa.R' 'check-metric.R' 'class-accuracy.R' 'class-bal_accuracy.R' 'class-detection_prevalence.R' 'class-f_meas.R' 'class-fall_out.R' 'class-j_index.R' 'class-kap.R' 'class-markedness.R' 'class-mcc.R' 'class-miss_rate.R' 'class-npv.R' 'class-ppv.R' 'class-precision.R'

'class-recall.R' 'class-roc_dist.R' 'class-sedi.R'
'class-sens.R' 'class-spec.R' 'conf_mat.R' 'data.R'
'deprecated-prob_helpers.R' 'deprecated-template.R'
'estimator-helpers.R' 'event-level.R' 'fair-aaa.R'
'fair-demographic_parity.R' 'fair-equal_opportunity.R'
'fair-equalized_odds.R' 'import-standalone-obj-type.R'
'import-standalone-survival.R' 'metric-tweak.R'
'metric-type-docs.R' 'misc.R' 'missings.R' 'num-ccc.R'
'num-gini_coef.R' 'num-huber_loss.R' 'num-huber_loss_pseudo.R'
'num-iic.R' 'num-mae.R' 'num-mape.R' 'num-mase.R' 'num-mpe.R'
'num-msd.R' 'num-mse.R' 'num-poisson_log_loss.R' 'num-rmse.R'
'num-rmse_relative.R' 'num-rpd.R' 'num-rpiq.R' 'num-rsq.R'
'num-rsq_trad.R' 'num-smape.R'
'orderedprob-ranked_prob_score.R' 'prob-average_precision.R'
'prob-binary-thresholds.R' 'prob-brier_class.R'
'prob-classification_cost.R' 'prob-gain_capture.R'
'prob-helpers.R' 'prob-mn_log_loss.R' 'prob-pr_auc.R'
'prob-roc_auc.R' 'prob-roc_aunp.R' 'prob-roc_aunu.R'
'probcurve-gain_curve.R' 'probcurve-lift_curve.R'
'probcurve-pr_curve.R' 'probcurve-roc_curve.R'
'quant-weighted_interval_score.R' 'reexports.R'
'surv-brier_survival.R' 'surv-brier_survival_integrated.R'
'surv-concordance_survival.R' 'surv-roc_auc_survival.R'
'surv-roc_curve_survival.R' 'surv-royston.R' 'template.R'
'validation.R' 'yardstick-package.R'

NeedsCompilation yes

Author Max Kuhn [aut],
Davis Vaughan [aut],
Emil Hvitfeldt [aut, cre] (ORCID:
<https://orcid.org/0000-0002-0679-1945>),
Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Emil Hvitfeldt <emil.hvitfeldt@posit.co>

Repository CRAN

Date/Publication 2026-04-07 17:10:02 UTC

Contents

`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, case_weights = NULL, ...)

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

Arguments

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

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

Details

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formula used here is:

$$\text{Accuracy} = \frac{A + D}{A + B + C + D}$$

Accuracy is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect predictions.

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

All class metrics

Other class metrics: bal_accuracy(), detection_prevalence(), f_meas(), fall_out(), j_index(), kap(), markedness(), mcc(), miss_rate(), npv(), ppv(), precision(), recall(), roc_dist(), sedi(), sens(), spec()

Examples

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

# Two class
accuracy(two_class_example, truth, predicted)

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

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

---

average_precision *Area under the precision recall curve*

Description

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

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

Usage

```
average_precision(data, ...)

## S3 method for class 'data.frame'
average_precision(
  data,
  truth,
  ...,
  estimator = NULL,
```

```
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL
)

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

Arguments

data	A data.frame containing the columns specified by truth and ....
...	A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimator	One of "binary", "macro", or "macro_weighted" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other two are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on truth.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
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

Average precision is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect precision and recall at all thresholds.

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

$$AP = \sum (r_i - r_{i-1}) \cdot p_i$$

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

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

Value

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

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

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

Multiclass

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

Relevant Level

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

See Also

All probability metrics

`pr_curve()` for computing the full precision recall curve.

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

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

Examples

```
# ------------------------------------------------------------------------------
# Two class example

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

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

# ------------------------------------------------------------------------------
# Multiclass example

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

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

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

# Groups are respected
hpc_cv |>
  group_by(Resample) |>
  average_precision(obs, VF:L)

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

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

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

---

bal_accuracy *Balanced accuracy*

Description

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

Usage

```
bal_accuracy(data, ...)

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

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

Arguments

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

truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

Details

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

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

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

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2}$$

Balanced accuracy is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect predictions.

Value

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

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

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

Relevant Level

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

Multiclass

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

Author(s)

Max Kuhn

See Also

All class metrics

Other class metrics: `accuracy()`, `detection_prevalence()`, `f_meas()`, `fall_out()`, `j_index()`, `kap()`, `markedness()`, `mcc()`, `miss_rate()`, `npv()`, `ppv()`, `precision()`, `recall()`, `roc_dist()`, `sedi()`, `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
bal_accuracy_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)
```

`brier_class` *Brier score for classification models*

Description

Compute the Brier score for a classification model.

Usage

```
brier_class(data, ...)

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

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

Arguments

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

Otherwise, there should be as many columns as factor levels of `truth` and the ordering of the columns should be the same as the factor levels of `truth`.

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

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

`event_level` A single string. Either `"first"` or `"second"` to specify which level of `truth` to consider as the "event". This argument is only applicable when `estimator = "binary"`. The default uses an internal helper that defaults to `"first"`.

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

`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

Brier score is a metric that should be minimized. The output ranges from 0 to 1, with 0 indicating perfect predictions.

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

The formula used here is:

$$\text{Brier} = \frac{1}{2N} \sum_{i=1}^{N} \sum_{j=1}^{K} (y_{ij} - p_{ij})^2$$

where N is the number of observations, K is the number of classes, y_{ij} is 1 if observation i belongs to class j and 0 otherwise, and p_{ij} is the predicted probability of observation i for class j.

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

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

Value

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

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

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

Multiclass

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

Relevant Level

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

Author(s)

Max Kuhn

References

Kruppa, J., Liu, Y., Diener, H.-C., Holste, T., Weimar, C., Koonig, I. R., and Ziegler, A. (2014) Probability estimation with machine learning methods for dichotomous and multicategory outcome: Applications. Biometrical Journal, 56 (4): 564-583.

See Also

All probability metrics

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

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

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

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

`brier_survival` *Time-Dependent Brier score for right censored data*

Description

Compute the time-dependent Brier score for right censored data, which is the mean squared error at time point `.eval_time`.

Usage

```
brier_survival(data, ...)

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

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

Arguments

`data`	A `data.frame` containing the columns specified by `truth` and `....`
`...`	The column identifier for the survival probabilities this should be a list column of data.frames corresponding to the output given when predicting with censored model. 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, the dots are not used.
`truth`	The column identifier for the true survival result (that is created using `survival::Surv()`.). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, an `survival::Surv()` object.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`case_weights`	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.
`estimate`	A list column of data.frames corresponding to the output given when predicting with censored model. See the details for more information regarding format.

Details

Brier survival score is a metric that should be minimized. The output ranges from 0 to 1, with 0 indicating perfect predictions.

This formulation takes survival probability predictions at one or more specific *evaluation times* and, for each time, computes the Brier score. To account for censoring, inverse probability of censoring weights (IPCW) are used in the calculations.

The column passed to `...` should be a list column with one element per independent experiential unit (e.g. patient). The list column should contain data frames with several columns:

- `.eval_time`: The time that the prediction is made.
- `.pred_survival`: The predicted probability of survival up to `.eval_time`
- `.weight_censored`: The case weight for the inverse probability of censoring.

The last column can be produced using `parsnip::.censoring_weights_graf()`. This corresponds to the weighting scheme of Graf *et al* (1999). The internal data set `lung_surv` shows an example of the format.

This method automatically groups by the `.eval_time` argument.

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

Value

A `tibble` with columns `.metric`, `.estimator`, and `.estimate`.

For an ungrouped data frame, the result has one row of values. For a grouped data frame, the number of rows returned is the same as the number of groups.

For `brier_survival_vec()`, a `numeric` vector same length as the input argument `eval_time`. (or `NA`).

Author(s)

Emil Hvitfeldt

References

E. Graf, C. Schmoor, W. Sauerbrei, and M. Schumacher, "Assessment and comparison of prognostic classification schemes for survival data," *Statistics in Medicine*, vol. 18, no. 17-18, pp. 2529–2545, 1999.

See Also

All dynamic survival metrics

Other dynamic survival metrics: `brier_survival_integrated()`, `roc_auc_survival()`

Examples

```
# example code

library(dplyr)

lung_surv |>
  brier_survival(
    truth = surv_obj,
    .pred
  )
```

`brier_survival_integrated` *Integrated Brier score for right censored data*

Description

Compute the integrated Brier score for right censored data, which is an overall calculation of model performance for all values of `.eval_time`.

Usage

```
brier_survival_integrated(data, ...)

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

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

Arguments

`data`	A `data.frame` containing the columns specified by `truth` and `....`
`...`	The column identifier for the survival probabilities this should be a list column of data.frames corresponding to the output given when predicting with censored model. 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, the dots are not used.
`truth`	The column identifier for the true survival result (that is created using `survival::Surv()`.). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, an `survival::Surv()` object.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`case_weights`	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.
`estimate`	A list column of data.frames corresponding to the output given when predicting with censored model. See the details for more information regarding format.

Details

Integrated Brier survival score is a metric that should be minimized. The output ranges from 0 to 1, with 0 indicating perfect predictions.

The integrated time-dependent brier score is calculated in an "area under the curve" fashion. The brier score is calculated for each value of `.eval_time`. The area is calculated via the trapezoidal rule. The area is divided by the largest value of `.eval_time` to bring it into the same scale as the traditional brier score.

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

This formulation takes survival probability predictions at one or more specific *evaluation times* and, for each time, computes the Brier score. To account for censoring, inverse probability of censoring weights (IPCW) are used in the calculations.

The column passed to `...` should be a list column with one element per independent experiential unit (e.g. patient). The list column should contain data frames with several columns:

- `.eval_time`: The time that the prediction is made.
- `.pred_survival`: The predicted probability of survival up to `.eval_time`
- `.weight_censored`: The case weight for the inverse probability of censoring.

The last column can be produced using `parsnip::.censoring_weights_graf()`. This corresponds to the weighting scheme of Graf *et al* (1999). The internal data set `lung_surv` shows an example of the format.

This method automatically groups by the `.eval_time` argument.

Value

A `tibble` with columns `.metric`, `.estimator`, and `.estimate`.

For an ungrouped data frame, the result has one row of values. For a grouped data frame, the number of rows returned is the same as the number of groups.

For `brier_survival_integrated_vec()`, a `numeric` vector same length as the input argument `eval_time`. (or `NA`).

Author(s)

Emil Hvitfeldt

References

E. Graf, C. Schmoor, W. Sauerbrei, and M. Schumacher, "Assessment and comparison of prognostic classification schemes for survival data," Statistics in Medicine, vol. 18, no. 17-18, pp. 2529–2545, 1999.

See Also

All integrated survival metrics

Other dynamic survival metrics: `brier_survival()`, `roc_auc_survival()`

Examples

```
library(dplyr)

lung_surv |>
  brier_survival_integrated(
    truth = surv_obj,
    .pred
  )
```

---

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,
  case_weights = NULL,
  ...
)

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

Arguments

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

bias	A logical; should the biased estimate of variance be used (as is Lin (1989))?
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().

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

CCC is a metric that should be maximized. The output ranges from -1 to 1, with 1 indicating perfect agreement.

The formula for CCC is:

$$\text{CCC} = \frac{2 \cdot \text{cov}(\text{truth}, \text{estimate})}{\text{var}(\text{truth}) + \text{var}(\text{estimate}) + (\bar{\text{truth}} - \bar{\text{estimate}})^2}$$

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

All numeric metrics

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

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

Other accuracy metrics: huber_loss(), huber_loss_pseudo(), iic(), mae(), mape(), mase(), mpe(), msd(), mse(), poisson_log_loss(), rmse(), rmse_relative(), 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))
# Using a different value of 'bias'... if you are adding the metric to a
# metric set, you can create a new metric function with the updated argument
# value:

ccc_bias <- metric_tweak("ccc_bias", ccc, bias = TRUE)
multi_metrics <- metric_set(ccc, ccc_bias)
multi_metrics(solubility_test, solubility, prediction)
```

`check_metric` *Developer function for checking inputs in new metrics*

Description

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

Usage

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

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

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

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

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

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

check_linear_pred_survival_metric(
  truth,
  estimate,
  case_weights,
  call = caller_env()
)
```

```
check_quantile_metric(truth, estimate, case_weights, call = caller_env())
```

Arguments

truth	The realized vector of truth. • For check_numeric_metric(), a numeric vector. • For check_class_metric(), a factor. • For check_prob_metric(), a factor. • For check_ordered_prob_metric(), an ordered factor. • For check_dynamic_survival_metric(), a Surv object. • For check_static_survival_metric(), a Surv object. • For check_quantile_metric(), a numeric vector.
estimate	The realized estimate result. • For check_numeric_metric(), a numeric vector. • For check_class_metric(), a factor. • For check_prob_metric(), a numeric vector for binary truth, a numeric matrix for multic-class truth. • For check_ordered_prob_metric(), a numeric vector for binary truth, a numeric matrix for multic-class truth. • For check_dynamic_survival_metric(), list-column of data.frames. • For check_static_survival_metric(), a numeric vector. • For check_quantile_metric(), a hardhat::quantile_pred vector.
case_weights	The realized case weights, as a numeric vector. This must be the same length as truth.
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
estimator	This can either be NULL for the default auto-selection of averaging ("binary" or "macro"), or a single character to pass along to the metric implementation describing the kind of averaging to use.

See Also

metric-summarizers

---

classification_cost *Costs function for poor classification*

Description

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

Usage

```
classification_cost(data, ...)

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

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

Arguments

data	A `data.frame` containing the columns specified by `truth` and `...`.
...	A set of unquoted column names or one or more `dplyr` selector functions to choose which variables contain the class probabilities. If `truth` is binary, only 1 column should be selected, and it should correspond to the value of `event_level`. Otherwise, there should be as many columns as factor levels of `truth` and the ordering of the columns should be the same as the factor levels of `truth`.
truth	The column identifier for the true class results (that is a `factor`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `factor` vector.
costs	A data frame with columns `"truth"`, `"estimate"`, and `"cost"`. `"truth"` and `"estimate"` should be character columns containing unique combinations of the levels of the `truth` factor. `"cost"` should be a numeric column representing the cost that should be applied when the `"estimate"` is predicted, but the true result is `"truth"`. It is often the case that when `"truth" == "estimate"`, the cost is zero (no penalty for correct predictions). If any combinations of the levels of `truth` are missing, their costs are assumed to be zero. If `NULL`, equal costs are used, applying a cost of `0` to correct predictions, and a cost of `1` to incorrect predictions.

na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
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

Classification cost is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions (when costs for correct predictions are zero).

The formula used here is:

$$\text{Cost} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} c_{t_i,j} \cdot p_{ij}$$

where N is the number of observations, K is the number of classes, $c_{t_i,j}$ is the cost of predicting class j when the true class is t_i, and p_{ij} is the predicted probability of observation i for class j.

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

Value

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

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

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

Author(s)

Max Kuhn

See Also

All probability metrics

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

Examples

```
library(dplyr)

# ------------------------------------------------------------------------------
# Two class example
data(two_class_example)

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

# Assuming `Class1` is our "event", this penalizes false negatives heavily
costs2 <- tribble(
  ~truth,   ~estimate, ~cost,
  "Class1", "Class2",  2,
  "Class2", "Class1",  1
)

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

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

# ------------------------------------------------------------------------------
# Multiclass
data(hpc_cv)

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

# You can use the col1:colN tidyselect syntax
hpc_cv |>
```

```
  filter(Resample == "Fold01") |>
  classification_cost(obs, VF:L, costs = hpc_costs)

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

# If this metric will be used in a metric set, you can create a new metric
# function with the updated argument value:

class_costs  <- metric_tweak("class_costs", classification_cost, costs = hpc_costs)
multi_metrics <- metric_set(class_costs, roc_auc)
multi_metrics(hpc_cv, obs, VF:L)
```

`concordance_survival` *Concordance index for right-censored data*

Description

Compute the Concordance index for right-censored data

Usage

```
concordance_survival(data, ...)

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

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

Arguments

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

...	Not currently used.
truth	The column identifier for the true survival result (that is created using survival::Surv().). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, an survival::Surv() object.
estimate	The column identifier for the predicted time, this should be a numeric variables. 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.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().

Details

Concordance is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect concordance.

The concordance index is defined as the proportion of all comparable pairs in which the predictions and outcomes are concordant.

Two observations are comparable if:

1. both of the observations experienced an event (at different times), or
2. the observation with the shorter observed survival time experienced an event, in which case the event-free subject "outlived" the other.

A pair is not comparable if they experienced events at the same time.

Concordance intuitively means that two samples were ordered correctly by the model. More specifically, two samples are concordant, if the one with a higher estimated risk score has a shorter actual survival time.

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

Value

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

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

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

Author(s)

Emil Hvitfeldt

References

Harrell, F.E., Califf, R.M., Pryor, D.B., Lee, K.L., Rosati, R.A, "Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors", Statistics in Medicine, 15(4), 361-87, 1996.

See Also

All static survival metrics

Examples

```
concordance_survival(
  data = lung_surv,
  truth = surv_obj,
  estimate = .pred_time
)
```

---

conf_mat *Confusion Matrix for Categorical Data*

Description

Calculates a cross-tabulation of observed and predicted classes.

Usage

```
conf_mat(data, ...)

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

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

Arguments

data	A data frame or a base::table().
...	Not used.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.

dnn	A character vector of dimnames for the table.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
x	A conf_mat object.

Details

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

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

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

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

Value

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

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

See Also

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

Examples

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

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

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

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

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

counts_per_resample

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

round(mean_cmat, 3)

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

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

# And can be modified further like normal ggplot2 objects
autoplot(cm, type = "heatmap") +
  geom_label(aes(label = Freq), fill = "white")
```

`demographic_parity` *Demographic parity*

Description

Demographic parity is satisfied when a model's predictions have the same predicted positive rate across groups. A value of 0 indicates parity across groups. Note that this definition does not depend on the true outcome; the `truth` argument is included in outputted metrics for consistency.

`demographic_parity()` is calculated as the difference between the largest and smallest value of `detection_prevalence()` across groups.

Demographic parity is sometimes referred to as group fairness, disparate impact, or statistical parity.

See the "Measuring Disparity" section for details on implementation.

Usage

```
demographic_parity(by)
```

Arguments

by	The column identifier for the sensitive feature. This should be an unquoted column name referring to a column in the un-preprocessed data.

Value

This function outputs a yardstick *fairness metric* function. Given a grouping variable by, `demographic_parity()` will return a yardstick metric function that is associated with the data-variable grouping by and a post-processor. The outputted function will first generate a set of detection_prevalence metric values by group before summarizing across groups using the post-processing function.

The outputted function only has a data frame method and is intended to be used as part of a metric set.

Measuring Disparity

By default, this function takes the difference in range of detection_prevalence `.estimates` across groups. That is, the maximum pair-wise disparity between groups is the return value of `demographic_parity()`'s `.estimate`.

For finer control of group treatment, construct a context-aware fairness metric with the `new_groupwise_metric()` function by passing a custom `aggregate` function:

```
# the actual default `aggregate` is:
diff_range <- function(x, ...) {diff(range(x$.estimate))}

demographic_parity_2 <-
  new_groupwise_metric(
    fn = detection_prevalence,
    name = "demographic_parity_2",
    aggregate = diff_range
  )
```

In `aggregate()`, `x` is the `metric_set()` output with detection_prevalence values for each group, and `...` gives additional arguments (such as a grouping level to refer to as the "baseline") to pass to the function outputted by `demographic_parity_2()` for context.

References

Agarwal, A., Beygelzimer, A., Dudik, M., Langford, J., & Wallach, H. (2018). "A Reductions Approach to Fair Classification." Proceedings of the 35th International Conference on Machine Learning, in Proceedings of Machine Learning Research. 80:60-69.

Verma, S., & Rubin, J. (2018). "Fairness definitions explained". In Proceedings of the international workshop on software fairness (pp. 1-7).

Bird, S., Dudík, M., Edgar, R., Horn, B., Lutz, R., Milan, V., ... & Walker, K. (2020). "Fairlearn: A toolkit for assessing and improving fairness in AI". Microsoft, Tech. Rep. MSR-TR-2020-32.

See Also

Other fairness metrics: `equal_opportunity()`, `equalized_odds()`

Examples

```
library(dplyr)

data(hpc_cv)

head(hpc_cv)

# evaluate `demographic_parity()` by Resample
m_set <- metric_set(demographic_parity(Resample))

# use output like any other metric set
hpc_cv |>
  m_set(truth = obs, estimate = pred)

# can mix fairness metrics and regular metrics
m_set_2 <- metric_set(sens, demographic_parity(Resample))

hpc_cv |>
  m_set_2(truth = obs, estimate = pred)
```

---

detection_prevalence *Detection prevalence*

Description

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

Usage

```
detection_prevalence(data, ...)

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

detection_prevalence_vec(
  truth,
  estimate,
  estimator = NULL,
```

```
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)
```

Arguments

`data`	Either a `data.frame` containing the columns specified by the `truth` and `estimate` arguments, or a `table/matrix` where the true class results should be in the columns of the table.
`...`	Not currently used.
`truth`	The column identifier for the true class results (that is a `factor`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `factor` vector.
`estimate`	The column identifier for the predicted class results (that is also `factor`). As with `truth` this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a `factor` vector.
`estimator`	One of: `"binary"`, `"macro"`, `"macro_weighted"`, or `"micro"` to specify the type of averaging to be done. `"binary"` is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose `"binary"` or `"macro"` based on `estimate`.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`case_weights`	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.
`event_level`	A single string. Either `"first"` or `"second"` to specify which level of `truth` to consider as the "event". This argument is only applicable when `estimator = "binary"`. The default uses an internal helper that defaults to `"first"`.

Details

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formula used here is:

$$\text{Detection Prevalence} = \frac{A + B}{A + B + C + D}$$

Detection prevalence is a metric that should be maximized. The output ranges from 0 to 1. The "optimal" value depends on the true prevalence of positive events in the data.

Value

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

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

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

Relevant Level

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

Multiclass

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

Author(s)

Max Kuhn

See Also

All class metrics

Other class metrics: `accuracy()`, `bal_accuracy()`, `f_meas()`, `fall_out()`, `j_index()`, `kap()`, `markedness()`, `mcc()`, `miss_rate()`, `npv()`, `ppv()`, `precision()`, `recall()`, `roc_dist()`, `sedi()`, `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
detection_prevalence_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)
```

---

developer-helpers *Developer helpers*

Description

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

Usage

```
dots_to_estimate(data, ...)

get_weights(data, estimator)

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

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

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

Arguments

data	A table with truth values as columns and predicted values as rows.
...	A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.
estimator	Either NULL for auto-selection, or a single character for the type of estimator to use.
x	The column used to autoselect the estimator. This is generally the truth column, but can also be a table if your metric has table methods.
metric_class	A single character of the name of the metric to autoselect the estimator for. This should match the method name created for finalize_estimator_internal().
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
metric_dispatcher	A simple dummy object with the class provided to metric_class. This is created and passed along for you.
estimator_override	A character vector overriding the default allowed estimator list of c("binary", "macro", "micro", "macro_weighted"). Set this if your classification estimator does not support all of these methods.

Dots -> Estimate

[Deprecated]

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

Weight Calculation

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

Estimator Selection

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

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

`finalize_estimator_internal.metric_name`. Your method for `finalize_estimator_internal()` should do two things:

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

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

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

Estimator Validation

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

See Also

metric-summarizers check_metric yardstick_remove_missing

`equalized_odds` *Equalized odds*

Description

Equalized odds is satisfied when a model's predictions have the same false positive, true positive, false negative, and true negative rates across protected groups. A value of 0 indicates parity across groups.

By default, this function takes the maximum difference in range of `sens()` and `spec()` `.estimates` across groups. That is, the maximum pair-wise disparity in `sens()` or `spec()` between groups is the return value of `equalized_odds()`'s `.estimate`.

Equalized odds is sometimes referred to as conditional procedure accuracy equality or disparate mistreatment.

See the "Measuring disparity" section for details on implementation.

Usage

```
equalized_odds(by)
```

Arguments

by	The column identifier for the sensitive feature. This should be an unquoted column name referring to a column in the un-preprocessed data.

Value

This function outputs a yardstick *fairness metric* function. Given a grouping variable by, `equalized_odds()` will return a yardstick metric function that is associated with the data-variable grouping by and a post-processor. The outputted function will first generate a set of `sens()` and `spec()` metric values by group before summarizing across groups using the post-processing function.

The outputted function only has a data frame method and is intended to be used as part of a metric set.

Measuring Disparity

For finer control of group treatment, construct a context-aware fairness metric with the `new_groupwise_metric()` function by passing a custom `aggregate` function:

```
# see yardstick:::max_positive_rate_diff for the actual `aggregate()`
diff_range <- function(x, ...) {diff(range(x$.estimate))}

equalized_odds_2 <-
  new_groupwise_metric(
    fn = metric_set(sens, spec),
    name = "equalized_odds_2",
    aggregate = diff_range
  )
```

In `aggregate()`, `x` is the `metric_set()` output with `sens()` and `spec()` values for each group, and `...` gives additional arguments (such as a grouping level to refer to as the "baseline") to pass to the function outputted by `equalized_odds_2()` for context.

References

Agarwal, A., Beygelzimer, A., Dudik, M., Langford, J., & Wallach, H. (2018). "A Reductions Approach to Fair Classification." Proceedings of the 35th International Conference on Machine Learning, in Proceedings of Machine Learning Research. 80:60-69.

Verma, S., & Rubin, J. (2018). "Fairness definitions explained". In Proceedings of the international workshop on software fairness (pp. 1-7).

Bird, S., Dudík, M., Edgar, R., Horn, B., Lutz, R., Milan, V., ... & Walker, K. (2020). "Fairlearn: A toolkit for assessing and improving fairness in AI". Microsoft, Tech. Rep. MSR-TR-2020-32.

See Also

Other fairness metrics: `demographic_parity()`, `equal_opportunity()`

Examples

```
library(dplyr)

data(hpc_cv)

head(hpc_cv)

# evaluate `equalized_odds()` by Resample
m_set <- metric_set(equalized_odds(Resample))

# use output like any other metric set
hpc_cv |>
  m_set(truth = obs, estimate = pred)

# can mix fairness metrics and regular metrics
m_set_2 <- metric_set(sens, equalized_odds(Resample))

hpc_cv |>
  m_set_2(truth = obs, estimate = pred)
```

---

equal_opportunity *Equal opportunity*

Description

Equal opportunity is satisfied when a model's predictions have the same true positive and false negative rates across protected groups. A value of 0 indicates parity across groups.

`equal_opportunity()` is calculated as the difference between the largest and smallest value of `sens()` across groups.

Equal opportunity is sometimes referred to as equality of opportunity.

See the "Measuring Disparity" section for details on implementation.

Usage

```
equal_opportunity(by)
```

Arguments

by	The column identifier for the sensitive feature. This should be an unquoted column name referring to a column in the un-preprocessed data.

Value

This function outputs a yardstick *fairness metric* function. Given a grouping variable `by`, `equal_opportunity()` will return a yardstick metric function that is associated with the data-variable grouping `by` and a post-processor. The outputted function will first generate a set of sens metric values by group before summarizing across groups using the post-processing function.

The outputted function only has a data frame method and is intended to be used as part of a metric set.

Measuring Disparity

By default, this function takes the difference in range of sens `.estimates` across groups. That is, the maximum pair-wise disparity between groups is the return value of `equal_opportunity()`'s `.estimate`.

For finer control of group treatment, construct a context-aware fairness metric with the `new_groupwise_metric()` function by passing a custom `aggregate` function:

```
# the actual default `aggregate` is:
diff_range <- function(x, ...) {diff(range(x$.estimate))}

equal_opportunity_2 <-
  new_groupwise_metric(
    fn = sens,
    name = "equal_opportunity_2",
    aggregate = diff_range
  )
```

In `aggregate()`, `x` is the `metric_set()` output with sens values for each group, and `...` gives additional arguments (such as a grouping level to refer to as the "baseline") to pass to the function outputted by `equal_opportunity_2()` for context.

References

Hardt, M., Price, E., & Srebro, N. (2016). "Equality of opportunity in supervised learning". Advances in neural information processing systems, 29.

Verma, S., & Rubin, J. (2018). "Fairness definitions explained". In Proceedings of the international workshop on software fairness (pp. 1-7).

Bird, S., Dudík, M., Edgar, R., Horn, B., Lutz, R., Milan, V., ... & Walker, K. (2020). "Fairlearn: A toolkit for assessing and improving fairness in AI". Microsoft, Tech. Rep. MSR-TR-2020-32.

See Also

Other fairness metrics: `demographic_parity()`, `equalized_odds()`

Examples

```
library(dplyr)

data(hpc_cv)

head(hpc_cv)

# evaluate `equal_opportunity()` by Resample
m_set <- metric_set(equal_opportunity(Resample))

# use output like any other metric set
hpc_cv |>
  m_set(truth = obs, estimate = pred)
```

```
# can mix fairness metrics and regular metrics
m_set_2 <- metric_set(sens, equal_opportunity(Resample))

hpc_cv |>
  m_set_2(truth = obs, estimate = pred)
```

`fall_out` *Fall-out (False Positive Rate)*

Description

These functions calculate the fall-out (false positive rate) of a measurement system compared to a reference result (the "truth" or gold standard). Fall-out is defined as `1 - specificity`, or equivalently, the proportion of negatives that are incorrectly classified as positives.

Usage

```
fall_out(data, ...)

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

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

Arguments

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

truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

Details

Fall-out is also known as the false positive rate (FPR) or the probability of false alarm.

When the denominator of the calculation is 0, fall-out is undefined. This happens when both # true_negative = 0 and # false_positive = 0 are true, which means that there were no negatives. When computing binary fall-out, a NA value will be returned with a warning. When computing multiclass fall-out, the individual NA values will be removed, and the computation will proceed, with a warning.

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formula used here is:

$$\text{Fall-out} = \frac{B}{B + D}$$

Fall-out is a metric that should be minimized. The output ranges from 0 to 1, with 0 indicating that all actual negatives were correctly predicted as negative (no false positives).

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 `fall_out_vec()`, a single `numeric` value (or `NA`).

Relevant Level

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

Multiclass

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

See Also

All class metrics

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

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

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv |>
  filter(Resample == "Fold01") |>
  fall_out(obs, pred)

# Groups are respected
hpc_cv |>
  group_by(Resample) |>
  fall_out(obs, pred)

# Weighted macro averaging
hpc_cv |>
```

```
  group_by(Resample) |>
  fall_out(obs, pred, estimator = "macro_weighted")

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

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

`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,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

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

```
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)
```

Arguments

data	Either a `data.frame` containing the columns specified by the `truth` and `estimate` arguments, or a `table/matrix` where the true class results should be in the columns of the table.
...	Not currently used.
truth	The column identifier for the true class results (that is a `factor`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `factor` vector.
estimate	The column identifier for the predicted class results (that is also `factor`). As with `truth` this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a `factor` vector.
beta	A numeric value used to weight precision and recall. A value of 1 is traditionally used and corresponds to the harmonic mean of the two values but other values weight recall beta times more important than precision.
estimator	One of: `"binary"`, `"macro"`, `"macro_weighted"`, or `"micro"` to specify the type of averaging to be done. `"binary"` is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose `"binary"` or `"macro"` based on `estimate`.
na_rm	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.
event_level	A single string. Either `"first"` or `"second"` to specify which level of `truth` to consider as the "event". This argument is only applicable when `estimator = "binary"`. The default uses an internal helper that defaults to `"first"`.

Details

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

Suppose a 2x2 table with notation:

	Reference	
Predicted	Relevant	Irrelevant
Relevant	A	B
Irrelevant	C	D

The formulas used here are:

$$\text{Recall} = \frac{A}{A + C}$$

$$\text{Precision} = \frac{A}{A + B}$$

$$F_{meas} = \frac{(1 + \beta^2) \cdot \text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}}$$

F measure is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect precision and recall.

Value

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

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

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

Relevant Level

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

Multiclass

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

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

All class metrics

Other class metrics: accuracy(), bal_accuracy(), detection_prevalence(), fall_out(), j_index(), kap(), markedness(), mcc(), miss_rate(), npv(), ppv(), precision(), recall(), roc_dist(), sedi(), 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
f_meas_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)

# Using a different value of 'beta'... if you are adding the metric to a
# metric set, you can create a new metric function with the updated argument
# value:

f2_meas <- metric_tweak("f2_meas", f_meas, beta = 2)
multi_metrics <- metric_set(f_meas, f2_meas)
multi_metrics(two_class_example, truth, estimate = predicted)
```

gain_capture *Gain capture*

Description

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

Usage

```
gain_capture(data, ...)

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

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

Arguments

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

estimator	One of "binary", "macro", or "macro_weighted" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other two are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on truth.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
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.

Gain capture is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect discrimination.

The formula used here is:

$$\text{Gain Capture} = \frac{A_{model} - A_{baseline}}{A_{perfect} - A_{baseline}}$$

where A_{model} is the area under the model's gain curve, $A_{baseline}$ is the area under the baseline (random) gain curve, and $A_{perfect}$ is the area under a perfect gain curve. This is equivalent to twice the ROC AUC minus one, also known as the Gini coefficient.

Value

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

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

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

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the *first* level. To alter this, change the argument event_level to "second" to consider

the *last* level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Multiclass

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

Author(s)

Max Kuhn

References

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

All probability metrics

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

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

Examples

```
# ------------------------------------------------------------------------------
# Two class example

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

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

# ------------------------------------------------------------------------------
# Multiclass example

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

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

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

# Groups are respected
hpc_cv |>
  group_by(Resample) |>
  gain_capture(obs, VF:L)

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

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

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


# ------------------------------------------------------------------------------
# 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,
  event_level = yardstick_event_level(),
  case_weights = NULL
)
```

Arguments

data	A data.frame containing the columns specified by truth and ....
...	A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().

Details

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

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

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

Value

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

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

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

Gain and Lift Curves

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

The calculation to construct gain curves is as follows:

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

Multiclass

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

Relevant Level

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

Author(s)

Max Kuhn

References

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

```
# ------------------------------------------------------------------------------
# Two class example

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

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


# ------------------------------------------------------------------------------
# `autoplot()`

library(ggplot2)
library(dplyr)

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

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

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

`get_metrics` *Get all metrics of a given type*

Description

`get_metrics()` returns a `metric_set()` containing all yardstick metrics of the specified type(s).

Usage

```
get_metrics(type)
```

Arguments

`type`	A character vector of metric types. Valid types are: `"class"`, `"prob"`, `"ordered_prob"`, `"numeric"`, `"dynamic_survival"`, `"integrated_survival"`, `"static_survival"`, `"linear_pred_survival"`, and `"quantile"`. More than 1 type can be selected but you are constrained by which metric types `metric_set()` allows to be combined. This means that `get_metrics(c("class", "prob"))` will run without error, but `get_metrics(c("class", "numeric"))` will return an error because you can't combine `"class"` and `"numeric"` metrics.

Value

A `metric_set()` containing all metrics of the specified type(s).

See Also

`metric_set()`

Examples

```
get_metrics("numeric")

get_metrics("class")

# Get multiple types at once
get_metrics(c("class", "prob"))
```

gini_coef *Normalized Gini coefficient*

Description

Compute the normalized Gini coefficient, which measures the ranking ability of a regression model based on the Lorenz curve. This metric is useful for evaluating models that predict risk or loss costs, such as insurance pricing models.

Usage

```
gini_coef(data, ...)

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

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

Arguments

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

Details

The normalized Gini coefficient is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect ranking ability where predicted values perfectly rank the true values.

The Gini coefficient is calculated from the Lorenz curve, which plots the cumulative proportion of the total truth values against the cumulative proportion of observations when sorted by predicted values. The raw Gini is the area between the Lorenz curve and the diagonal line of equality. The normalized Gini divides this by the maximum possible Gini (achieved when observations are sorted by the true values).

The formula is:

$$\text{Normalized Gini} = \frac{G(\text{estimate})}{G(\text{truth})}$$

where $G(x)$ is the Gini coefficient when sorting by x.

Note that `gini_coef()` is a regression metric based on ranking, distinct from `gain_capture()` which is a classification metric.

Unlike many other metrics, `gini_coef()` is not symmetric with respect to `truth` and `estimate`. The `estimate` values determine the sorting order, while the `truth` values are accumulated along the Lorenz curve. Swapping them will produce different results.

When the true values are constant (zero variance), the Gini coefficient is undefined and `NA` is returned with a warning.

Value

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

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

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

See Also

All numeric metrics

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

Examples

```
# Supply truth and predictions as bare column names
gini_coef(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) |>
  gini_coef(solubility, prediction)

metric_results

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

---

hpc_cv *Multiclass Probability Predictions*

Description

Multiclass Probability Predictions

Details

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

Value

`hpc_cv` a data frame

Source

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

Examples

```
data(hpc_cv)
str(hpc_cv)

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

huber_loss *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,
  case_weights = NULL,
  ...
)

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

Arguments

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

delta	A single numeric value. Defines the boundary where the loss function transitions from quadratic to linear. Defaults to 1.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().

Details

Huber loss is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

The formula for Huber loss is:

$$L_\delta = \begin{cases} \frac{1}{2}a^2 & \text{if } |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

where $a = \text{truth}_i - \text{estimate}_i$.

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

All numeric metrics

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

Other accuracy metrics: ccc(), huber_loss_pseudo(), iic(), mae(), mape(), mase(), mpe(), msd(), mse(), poisson_log_loss(), rmse(), rmse_relative(), 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))
# Using a different value of 'delta'... if you are adding the metric to a
# metric set, you can create a new metric function with the updated argument
# value:

huber_loss_2 <- metric_tweak("huber_loss_2", huber_loss, delta = 2)
multi_metrics <- metric_set(huber_loss, huber_loss_2)
multi_metrics(solubility_test, solubility, prediction)
```

---

huber_loss_pseudo *Psuedo-Huber Loss*

Description

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

Usage

```
huber_loss_pseudo(data, ...)

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

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

Arguments

data	A data.frame containing the columns specified by the truth and estimate arguments.
...	Not currently used.
truth	The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate	The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
delta	A single numeric value. Defines the boundary where the loss function transitions from quadratic to linear. Defaults to 1.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().

Details

Pseudo-Huber loss is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

The formula for Pseudo-Huber loss is:

$$L_\delta = \frac{1}{n}\sum_{i=1}^{n} \delta^2 \left(\sqrt{1 + \left(\frac{\text{truth}_i - \text{estimate}_i}{\delta} \right)^2} - 1 \right)$$

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

All numeric metrics

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

Other accuracy metrics: `ccc()`, `huber_loss()`, `iic()`, `mae()`, `mape()`, `mase()`, `mpe()`, `msd()`, `mse()`, `poisson_log_loss()`, `rmse()`, `rmse_relative()`, `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))
# Using a different value of 'delta'... if you are adding the metric to a
# metric set, you can create a new metric function with the updated argument
# value:

huber_loss_pseudo_2 <- metric_tweak("huber_loss_pseudo_2", huber_loss_pseudo, delta = 2)
multi_metrics <- metric_set(huber_loss_pseudo, huber_loss_pseudo_2)
multi_metrics(solubility_test, solubility, prediction)
```

`iic` *Index of ideality of correlation*

Description

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

Note the application of IIC is useless under two conditions:

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

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

Usage

```
iic(data, ...)

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

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

Arguments

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

Details

IIC is a metric that should be maximized. The output ranges from -1 to 1, with 1 indicating perfect agreement.

The formula for IIC is:

$$\text{IIC} = \text{corr}(\text{truth}, \text{estimate}) \cdot \frac{\min(\text{MAE}^-, \text{MAE}^+)}{\max(\text{MAE}^-, \text{MAE}^+)}$$

where MAE^- and MAE^+ are the mean absolute errors for negative and non-negative residuals, respectively.

Value

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

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

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

Author(s)

Joyce Cahoon

References

Toropova, A. and Toropov, A. (2017). "The index of ideality of correlation. A criterion of predictability of QSAR models for skin permeability?" *Science of the Total Environment*. 586: 466-472.

See Also

All numeric metrics

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

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

Examples

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

library(dplyr)

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

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

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

metric_results

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

---

j_index *J-index*

Description

Youden's J statistic is defined as:

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,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

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

Arguments

data	Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
...	Not currently used.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().

`event_level` A single string. Either `"first"` or `"second"` to specify which level of `truth` to consider as the "event". This argument is only applicable when `estimator = "binary"`. The default uses an internal helper that defaults to `"first"`.

Details

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

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

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

$$\text{J-index} = \text{Sensitivity} + \text{Specificity} - 1$$

J-index is a metric that should be maximized. The output ranges from -1 to 1, with 1 indicating no false positives and no false negatives.

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

Value

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

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

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

Relevant Level

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

Multiclass

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

Author(s)

Max Kuhn

References

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

All class metrics

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

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv |>
  filter(Resample == "Fold01") |>
  j_index(obs, pred)

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

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

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

```
)

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

---

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,
  weighting = "none",
  na_rm = TRUE,
  case_weights = NULL,
  ...
)

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

Arguments

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

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

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

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

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

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

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

Details

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

$$p_o = \frac{A + D}{A + B + C + D}$$

$$p_e = \frac{(A + B)(A + C) + (C + D)(B + D)}{(A + B + C + D)^2}$$

$$\text{Kappa} = \frac{p_o - p_e}{1 - p_e}$$

where p_o is the observed agreement and p_e is the expected agreement by chance.

Kappa is a metric that should be maximized. The output ranges from -1 to 1, with 1 indicating perfect agreement. Negative values indicate agreement worse than chance.

Value

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

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

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

Multiclass

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

Author(s)

Max Kuhn

Jon Harmon

References

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

Cohen, J. (1968). "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit". *Psychological Bulletin*. 70 (4): 213-220.

See Also

All class metrics

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `fall_out()`, `j_index()`, `markedness()`, `mcc()`, `miss_rate()`, `npv()`, `ppv()`, `precision()`, `recall()`, `roc_dist()`, `sedi()`, `sens()`, `spec()`

Examples

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

# Two class
kap(two_class_example, truth, predicted)

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

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

```
# Using a different value of 'weighting'... if you are adding the metric to a
# metric set, you can create a new metric function with the updated argument
# value:

kap_lin  <- metric_tweak("kap_lin",  kap, weighting = "linear")
kap_quad <- metric_tweak("kap_quad", kap, weighting = "quadratic")
multi_metrics <- metric_set(kap, kap_lin, kap_quad)
multi_metrics(hpc_cv, obs, estimate = pred)
```

`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,
  event_level = yardstick_event_level(),
  case_weights = NULL
)
```

Arguments

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

event_level A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

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

Details

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

Value

A tibble with class lift_df or lift_grouped_df having columns:

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

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

Gain and Lift Curves

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

The calculation to construct lift curves is as follows:

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

Multiclass

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

Relevant Level

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

Author(s)

Max Kuhn

See Also

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

Examples

```
# ------------------------------------------------------------------------------
# Two class example

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

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


# ------------------------------------------------------------------------------
# `autoplot()`

library(ggplot2)
library(dplyr)

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

# Multiclass one-vs-all approach
# One curve per level
hpc_cv |>
```

```
  filter(Resample == "Fold01") |>
  lift_curve(obs, VF:L) |>
  autoplot()

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

---

lung_surv *Survival Analysis Results*

Description

Survival Analysis Results

Details

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

Value

`lung_surv` a data frame

Examples

```
data(lung_surv)
str(lung_surv)

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

---

mae *Mean absolute error*

Description

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

Usage

```
mae(data, ...)

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

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

Arguments

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

Details

MAE is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

The formula for MAE is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\text{truth}_i - \text{estimate}_i|$$

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

All numeric metrics

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

Other accuracy metrics: ccc(), huber_loss(), huber_loss_pseudo(), iic(), mape(), mase(), mpe(), msd(), mse(), poisson_log_loss(), rmse(), rmse_relative(), 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, case_weights = NULL, ...)

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

Arguments

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

Details

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

MAPE is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

The formula for MAPE is:

$$\text{MAPE} = \frac{100}{n} \sum_{i=1}^{n} \left| \frac{\text{truth}_i - \text{estimate}_i}{\text{truth}_i} \right|$$

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

All numeric metrics

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

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

`markedness` *Markedness*

Description

Markedness is defined as:

`precision()` + "inverse precision" - 1

where "inverse precision" is the proportion of true negatives among all predicted negatives. A related metric is Informedness, see the Details section for the relationship.

Usage

```
markedness(data, ...)

## S3 method for class 'data.frame'
markedness(
  data,
```

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

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

Arguments

data	Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
...	Not currently used.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

Details

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

$$\text{Precision} = \frac{A}{A + B}$$

$$\text{Inverse Precision} = \frac{D}{C + D}$$

$$\text{Markedness} = \text{Precision} + \text{Inverse Precision} - 1$$

Markedness is a metric that should be maximized. The output ranges from -1 to 1, with 1 indicating perfect predictions.

Markedness is to the predicted condition (precision and inverse precision) what Informedness (`j_index()`) is to the actual condition (sensitivity and specificity).

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 `markedness_vec()`, a single `numeric` value (or `NA`).

Relevant Level

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

Multiclass

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

References

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

All class metrics

Other class metrics: accuracy(), bal_accuracy(), detection_prevalence(), f_meas(), fall_out(), j_index(), kap(), mcc(), miss_rate(), npv(), ppv(), precision(), recall(), roc_dist(), sedi(), sens(), spec()

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv |>
  filter(Resample == "Fold01") |>
  markedness(obs, pred)

# Groups are respected
hpc_cv |>
  group_by(Resample) |>
  markedness(obs, pred)

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

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

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

---

mase *Mean absolute scaled error*

Description

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

Usage

```
mase(data, ...)

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

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

Arguments

`data`	A `data.frame` containing the columns specified by the `truth` and `estimate` arguments.
`...`	Not currently used.
`truth`	The column identifier for the true results (that is `numeric`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `numeric` vector.
`estimate`	The column identifier for the predicted results (that is also `numeric`). As with `truth` this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a `numeric` vector.
`m`	An integer value of the number of lags used to calculate the in-sample seasonal naive error. The default is used for non-seasonal time series. If each observation was at the daily level and the data showed weekly seasonality, then `m = 7L` would be a reasonable choice for a 7-day seasonal naive calculation.

`mae_train`	A numeric value which allows the user to provide the in-sample seasonal naive mean absolute error. If this value is not provided, then the out-of-sample seasonal naive mean absolute error will be calculated from `truth` and will be used instead.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`case_weights`	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.

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)`.

MASE is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

The formula for MASE is:

$$\text{MASE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|\text{truth}_i - \text{estimate}_i|}{\text{MAE}_{naive}}$$

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

All numeric metrics

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

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

Examples

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

library(dplyr)

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

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

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

metric_results

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

---

mcc *Matthews correlation coefficient*

Description

Matthews correlation coefficient

Usage

```
mcc(data, ...)

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

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

Arguments

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

Details

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formula used here is:

$$\text{MCC} = \frac{(A \cdot D) - (B \cdot C)}{\sqrt{(A+B)(A+C)(D+B)(D+C)}}$$

MCC is a metric that should be maximized. The output ranges from -1 to 1, with 1 indicating perfect predictions. A value of 0 indicates no better than random prediction, and negative values indicate inverse prediction.

Value

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

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

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

Relevant Level

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

Multiclass

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

Author(s)

Max Kuhn

References

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

See Also

All class metrics

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `fall_out()`, `j_index()`, `kap()`, `markedness()`, `miss_rate()`, `npv()`, `ppv()`, `precision()`, `recall()`, `roc_dist()`, `sedi()`, `sens()`, `spec()`

Examples

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

# Two class
mcc(two_class_example, truth, predicted)

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

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

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

Description

These functions are useful alongside check_metric and yardstick_remove_missing for implementing new custom metrics. These functions call the metric function inside `dplyr::summarise()` or `dplyr::reframe()` for `curve_metric_summarizer()`. See Custom performance metrics for more information.

Usage

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

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

prob_metric_summarizer(
```

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

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

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

dynamic_survival_metric_summarizer(
  name,
  fn,
  data,
  truth,
  ...,
  na_rm = TRUE,
```

```
  case_weights = NULL,
  fn_options = list(),
  error_call = caller_env()
)

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

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

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

quantile_metric_summarizer(
  name,
  fn,
  data,
  truth,
```

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

Arguments

name	A single character representing the name of the metric to use in the tibble output. This will be modified to include the type of averaging if appropriate.
fn	The vector version of your custom metric function. It generally takes truth, estimate, na_rm, and any other extra arguments needed to calculate the metric.
data	The data frame with truth and estimate columns passed in from the data frame version of your metric function that called the metric summarizer.
truth	The unquoted column name corresponding to the truth column.
estimate	Generally, the unquoted column name corresponding to the estimate column. For metrics that take multiple columns through ... like class probability metrics, this is a result of dots_to_estimate().
...	These dots are for future extensions and must be empty.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds. The removal is executed in yardstick_remove_missing().
case_weights	For metrics supporting case weights, an unquoted column name corresponding to case weights can be passed here. If not NULL, the case weights will be passed on to fn as the named argument case_weights.
fn_options	A named list of metric specific options. These are spliced into the metric function call using !!! from rlang. The default results in nothing being spliced into the call.
error_call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
estimator	This can either be NULL for the default auto-selection of averaging ("binary" or "macro"), or a single character to pass along to the metric implementation describing the kind of averaging to use.
event_level	This can either be NULL to use the default event_level value of the fn or a single string of either "first" or "second" to pass along describing which level should be considered the "event".

Details

The following functions are generally called from the data frame version of your metric function. They know how to call your metric over grouped data frames and return a tibble consistent with other metrics.

- numeric_metric_summarizer()

- class_metric_summarizer()
- prob_metric_summarizer()
- ordered_prob_metric_summarizer()
- curve_metric_summarizer()
- dynamic_survival_metric_summarizer()
- static_survival_metric_summarizer()
- curve_survival_metric_summarizer()
- linear_pred_survival_metric_summarizer()
- quantile_metric_summarizer()

See Also

check_metric yardstick_remove_missing finalize_estimator() dots_to_estimate()

---

metrics *General Function to Estimate Performance*

Description

This function estimates one or more common performance estimates depending on the class of truth (see **Value** below) and returns them in a three column tibble. If you wish to modify the metrics used or how they are used see metric_set().

Usage

```
metrics(data, ...)

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

Arguments

data	A data.frame containing the columns specified by truth, estimate, and ....
...	A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.
truth	The column identifier for the true results (that is numeric or factor). This should be an unquoted column name although this argument is passed by expression and support quasiquotation (you can unquote column names).
estimate	The column identifier for the predicted results (that is also numeric or factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name.

na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
options	[deprecated] No longer supported as of yardstick 1.0.0. If you pass something here it will be ignored with a warning. Previously, these were options passed on to pROC::roc(). If you need support for this, use the pROC package directly.

Value

A three column tibble.

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

See Also

metric_set(), get_metrics()

Examples

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

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

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

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

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

`metric_set` *Combine metric functions*

Description

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

Usage

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

Arguments

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

Details

All functions must be either:

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

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

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

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

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

```
  event_level = yardstick_event_level(),
  case_weights = NULL
)

# Dynamic / integrated / static survival metric set signature:
fn(
  data,
  truth,
  ...,
  estimate,
  na_rm = TRUE,
  case_weights = NULL
)
```

Naming the `estimate` argument

Note that for class/prob metric sets and survival metric sets, the `estimate` argument comes *after* `...` in the function signature. This means you must always pass `estimate` as a named argument; otherwise your column will be captured by `...` and you'll get an error.

```
# Correct - estimate is named:
my_metrics(two_class_example, truth, estimate = predicted)

# Incorrect - estimate is not named and gets captured by `...`:
my_metrics(two_class_example, truth, predicted)
```

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

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

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

See Also

`metrics()`, `get_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 using
# `metric_tweak()`. Here's an example where we use the `bias` option to the
# `ccc()` metric
ccc_with_bias <- metric_tweak("ccc_with_bias", ccc, bias = TRUE)

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_tweak` *Tweak a metric function*

Description

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

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

Usage

```
metric_tweak(.name, .fn, ...)
```

Arguments

`.name`	A single string giving the name of the new metric. This will be used in the `".metric"` column of the output.
`.fn`	An existing yardstick metric function to tweak.
`...`	Name-value pairs specifying which optional arguments to override and the values to replace them with. Arguments `data`, `truth`, and `estimate` are considered *protected*, and cannot be overridden, but all other optional arguments can be altered.

Details

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

Value

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

Examples

```
mase12 <- metric_tweak("mase12", mase, m = 12)

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

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

# This is most useful to set optional argument values ahead of time when
# using a metric set
mase10 <- metric_tweak("mase10", mase, m = 10)
metrics <- metric_set(mase, mase10, mase12)
metrics(solubility_test, solubility, prediction)
```

`miss_rate` *Miss rate (False Negative Rate)*

Description

These functions calculate the miss rate (false negative rate) of a measurement system compared to a reference result (the "truth" or gold standard). Miss rate is defined as `1 - sensitivity`, or equivalently, the proportion of positives that are incorrectly classified as negatives.

Usage

```
miss_rate(data, ...)

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

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

Arguments

data	Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
...	Not currently used.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().

`event_level` A single string. Either `"first"` or `"second"` to specify which level of `truth` to consider as the "event". This argument is only applicable when `estimator = "binary"`. The default uses an internal helper that defaults to `"first"`.

Details

Miss rate is also known as the false negative rate (FNR) or the probability of miss.

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

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formula used here is:

$$\text{Miss rate} = \frac{C}{A + C}$$

Miss rate is a metric that should be minimized. The output ranges from 0 to 1, with 0 indicating that all actual positives were correctly predicted as positive (no false negatives).

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 `miss_rate_vec()`, a single `numeric` value (or `NA`).

Relevant Level

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

Multiclass

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

See Also

All class metrics

Other class metrics: accuracy(), bal_accuracy(), detection_prevalence(), f_meas(), fall_out(), j_index(), kap(), markedness(), mcc(), npv(), ppv(), precision(), recall(), roc_dist(), sedi(), sens(), spec()

Other sensitivity metrics: fall_out(), npv(), ppv(), sens(), spec()

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv |>
  filter(Resample == "Fold01") |>
  miss_rate(obs, pred)

# Groups are respected
hpc_cv |>
  group_by(Resample) |>
  miss_rate(obs, pred)

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

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

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

---

mn_log_loss *Mean log loss for multinomial data*

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,
  event_level = yardstick_event_level(),
  case_weights = NULL
)

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

Arguments

data	A data.frame containing the columns specified by truth and ...
...	A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
sum	A logical. Should the sum of the likelihood contributions be returned (instead of the mean value)?
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

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

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 its result compared to the probability of .9.

Log loss is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

The formula used here is:

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij} \cdot \log(p_{ij})$$

where N is the number of observations, K is the number of classes, y_{ij} is 1 if observation i belongs to class j and 0 otherwise, and p_{ij} is the predicted probability of observation i for class j.

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

All probability metrics

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

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

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

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


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

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

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

---

mpe *Mean percentage error*

Description

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

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

Usage

```
mpe(data, ...)

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

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

Arguments

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

Details

MPE is a metric where the optimal value is 0. The output ranges from -Inf to Inf, with 0 indicating predictions are unbiased.

The formula for MPE is:

$$\text{MPE} = \frac{100}{n} \sum_{i=1}^{n} \frac{\text{truth}_i - \text{estimate}_i}{\text{truth}_i}$$

Using this convention, a *positive* MPE indicates under-prediction (on average, truth > estimate) and a *negative* MPE indicates over-prediction (on average, estimate > 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 mpe_vec(), a single numeric value (or NA).

Author(s)

Thomas Bierhance

See Also

All numeric metrics

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

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

Examples

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

mpe(solubility_test, solubility, prediction)

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

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

library(dplyr)

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

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

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

metric_results

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

msd *Mean signed deviation*

Description

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

Usage

```
msd(data, ...)

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

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

Arguments

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

Details

MSD is a metric where the optimal value is 0. The output ranges from -Inf to Inf, with 0 indicating predictions are unbiased.

The formula for MSD is:

$$\text{MSD} = \frac{1}{n}\sum_{i=1}^{n}(\text{truth}_i - \text{estimate}_i)$$

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

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

Value

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

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

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

Author(s)

Thomas Bierhance

See Also

All numeric metrics

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

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

Examples

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

library(dplyr)

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

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

# Compute the metric by group
```

```
metric_results <- solubility_resampled |>
  group_by(resample) |>
  msd(solubility, prediction)

metric_results

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

---

mse *Mean squared error*

Description

Calculate the mean squared error.

Usage

```
mse(data, ...)

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

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

Arguments

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

Details

MSE is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

The formula for MSE is:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\text{truth}_i - \text{estimate}_i)^2$$

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 `mse_vec()`, a single `numeric` value (or `NA`).

See Also

`rmse()` for the root mean squared error, which is the square root of MSE and is in the same units as the original data.

All numeric metrics

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

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

Examples

```
# Supply truth and predictions as bare column names
mse(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) |>
  mse(solubility, prediction)

metric_results

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

`new-metric` *Construct a new metric function*

Description

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

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

Usage

```
new_class_metric(fn, direction, range = NULL)

new_prob_metric(fn, direction, range = NULL)

new_ordered_prob_metric(fn, direction, range = NULL)

new_numeric_metric(fn, direction, range = NULL)

new_dynamic_survival_metric(fn, direction, range = NULL)

new_integrated_survival_metric(fn, direction, range = NULL)

new_static_survival_metric(fn, direction, range = NULL)

new_linear_pred_survival_metric(fn, direction, range = NULL)

new_quantile_metric(fn, direction, range = NULL)
```

Arguments

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

- "zero"

range — An optional numeric vector of length 2 giving the range of possible output values, e.g. c(0, 1) or c(0, Inf). Use -Inf and Inf for unbounded ranges.

---

new_groupwise_metric *Create groupwise metrics*

Description

Groupwise metrics quantify the disparity in value of a metric across a number of groups. Groupwise metrics with a value of zero indicate that the underlying metric is equal across groups. yardstick defines several common fairness metrics using this function, such as demographic_parity(), equal_opportunity(), and equalized_odds().

Usage

```
new_groupwise_metric(fn, name, aggregate, direction = "minimize")
```

Arguments

fn — A yardstick metric function or metric set.

name — The name of the metric to place in the .metric column of the output.

aggregate — A function to summarize the generated metric set results. The function takes metric set results as the first argument and returns a single numeric giving the .estimate value as output. See the Value and Examples sections for example uses.

direction — A string. One of:

- "maximize"
- "minimize"
- "zero"

Details

Note that *all* yardstick metrics are group-aware in that, when passed grouped data, they will return metric values calculated for each group. When passed grouped data, groupwise metrics also return metric values for each group, but those metric values are calculated by first additionally grouping by the variable passed to by and then summarizing the per-group metric estimates across groups using the function passed as the aggregate argument. Learn more about grouping behavior in yardstick using vignette("grouping", "yardstick").

Value

This function is a function factory; its output is itself a function. Further, the functions that this function outputs are also function factories. More explicitly, this looks like:

```
# a function with similar implementation to `demographic_parity()`:
diff_range <- function(x) {diff(range(x$.estimate))}

dem_parity <-
  new_groupwise_metric(
    fn = detection_prevalence,
    name = "dem_parity",
    aggregate = diff_range
  )
```

The outputted dem_parity is a function that takes one argument, by, indicating the data-masked variable giving the sensitive feature.

When called with a by argument, dem_parity will return a yardstick metric function like any other:

```
dem_parity_by_gender <- dem_parity(gender)
```

Note that dem_parity doesn't take any arguments other than by, and thus knows nothing about the data it will be applied to other than that it ought to have a column with name "gender" in it.

The output dem_parity_by_gender is a metric function that takes the same arguments as the function supplied as fn, in this case detection_prevalence. It will thus interface like any other yardstick function except that it will look for a "gender" column in the data it's supplied.

In addition to the examples below, see the documentation on the return value of fairness metrics like demographic_parity(), equal_opportunity(), or equalized_odds() to learn more about how the output of this function can be used.

Relevant Group Level

Additional arguments can be passed to the function outputted by the function that this function outputs. That is:

```
res_fairness <- new_groupwise_metric(...)
res_by <- res_fairness(by)
res_by(..., additional_arguments_to_aggregate = TRUE)
```

For finer control of how groups in by are treated, use the aggregate argument.

Examples

```
data(hpc_cv)

# `demographic_parity`, among other fairness metrics,
# is generated with `new_groupwise_metric()`:
diff_range <- function(x) {diff(range(x$.estimate))}
```

```
demographic_parity_ <-
  new_groupwise_metric(
    fn = detection_prevalence,
    name = "demographic_parity",
    aggregate = diff_range
  )

m_set <- metric_set(demographic_parity_(Resample))

m_set(hpc_cv, truth = obs, estimate = pred)

# the `post` argument can be used to accommodate a wide
# variety of parameterizations. to encode demographic
# parity as a ratio inside of a difference, for example:
ratio_range <- function(x, ...) {
  range <- range(x$.estimate)
  range[1] / range[2]
}

demographic_parity_ratio <-
  new_groupwise_metric(
    fn = detection_prevalence,
    name = "demographic_parity_ratio",
    aggregate = ratio_range
  )
```

---

npv *Negative predictive value*

Description

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

Usage

```
npv(data, ...)

## S3 method for class 'data.frame'
npv(
  data,
  truth,
  estimate,
  prevalence = NULL,
  estimator = NULL,
  na_rm = TRUE,
  case_weights = NULL,
```

```
  event_level = yardstick_event_level(),
  ...
)

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

Arguments

data	Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
...	Not currently used.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
prevalence	A numeric value for the rate of the "positive" class of the data.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

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.

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

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

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

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

$$\text{NPV} = \frac{\text{Specificity} \cdot (1 - \text{Prevalence})}{((1 - \text{Sensitivity}) \cdot \text{Prevalence}) + (\text{Specificity} \cdot (1 - \text{Prevalence}))}$$

NPV is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating all predicted negatives are true negatives.

Value

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

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

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

Relevant Level

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

Multiclass

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

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

All class metrics

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

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

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

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

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

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

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

# Making Class2 the "relevant" level
npv_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)
# Using a different value of 'prevalence'... if you are adding the metric to a
# metric set, you can create a new metric function with the updated argument
# value:
```

```
npv_alt_prev  <- metric_tweak("npv_alt_prev", npv, prevalence = 0.40)
multi_metrics <- metric_set(npv, npv_alt_prev)
multi_metrics(two_class_example, truth, estimate = predicted)
```

---

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)
```

---

poisson_log_loss *Mean log loss for Poisson data*

Description

Calculate the loss function for the Poisson distribution.

Usage

```
poisson_log_loss(data, ...)

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

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

Arguments

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

Details

Poisson log loss is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

The formula for Poisson log loss is:

$$L = \frac{1}{n}\sum_{i=1}^{n}(\log(\text{truth}_i!) + \text{estimate}_i - \text{truth}_i \cdot \log(\text{estimate}_i))$$

Value

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

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

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

Author(s)

Max Kuhn

See Also

All numeric metrics

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

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

Examples

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

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

---

ppv *Positive predictive value*

Description

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

Usage

```
ppv(data, ...)

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

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

Arguments

data	Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
...	Not currently used.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
prevalence	A numeric value for the rate of the "positive" class of the data.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

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.

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

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

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

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

$$\text{PPV} = \frac{\text{Sensitivity} \cdot \text{Prevalence}}{(\text{Sensitivity} \cdot \text{Prevalence}) + ((1 - \text{Specificity}) \cdot (1 - \text{Prevalence}))}$$

PPV is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating all predicted positives are true positives.

Value

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

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

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

Relevant Level

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

Multiclass

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

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

All class metrics

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

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

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv |>
  filter(Resample == "Fold01") |>
  ppv(obs, pred)

# Groups are respected
hpc_cv |>
  group_by(Resample) |>
  ppv(obs, pred)

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

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

# Making Class2 the "relevant" level
ppv_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)
# Using a different value of 'prevalence'... if you are adding the metric to a
# metric set, you can create a new metric function with the updated argument
# value:

ppv_alt_prev  <- metric_tweak("ppv_alt_prev", ppv, prevalence = 0.40)
multi_metrics <- metric_set(ppv, ppv_alt_prev)
multi_metrics(two_class_example, truth, estimate = predicted)

# 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,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

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

Arguments

data	Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
...	Not currently used.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

Details

The precision is the percentage of predicted truly relevant results of the total number of predicted relevant results and characterizes the "purity in retrieval performance" (Buckland and Gey, 1994).

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

Suppose a 2x2 table with notation:

	Reference	
Predicted	Relevant	Irrelevant
Relevant	A	B
Irrelevant	C	D

The formula used here is:

$$\text{Precision} = \frac{A}{A + B}$$

Precision is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating that all predicted positives were actual positives.

Value

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

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

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

Relevant Level

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

Multiclass

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

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

All class metrics

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `fall_out()`, `j_index()`, `kap()`, `markedness()`, `mcc()`, `miss_rate()`, `npv()`, `ppv()`, `recall()`, `roc_dist()`, `sedi()`, `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
precision_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)
```

---

pr_auc *Area under the precision recall curve*

Description

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

Usage

```
pr_auc(data, ...)

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

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

Arguments

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

`...`	A set of unquoted column names or one or more `dplyr` selector functions to choose which variables contain the class probabilities. If `truth` is binary, only 1 column should be selected, and it should correspond to the value of `event_level`. Otherwise, there should be as many columns as factor levels of `truth` and the ordering of the columns should be the same as the factor levels of `truth`.
`truth`	The column identifier for the true class results (that is a `factor`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `factor` vector.
`estimator`	One of `"binary"`, `"macro"`, or `"macro_weighted"` to specify the type of averaging to be done. `"binary"` is only relevant for the two class case. The other two are general methods for calculating multiclass metrics. The default will automatically choose `"binary"` or `"macro"` based on `truth`.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`event_level`	A single string. Either `"first"` or `"second"` to specify which level of `truth` to consider as the "event". This argument is only applicable when `estimator = "binary"`. The default uses an internal helper that defaults to `"first"`.
`case_weights`	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.
`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

PR AUC is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect precision and recall at all thresholds.

The area under the precision-recall curve is computed using the trapezoidal rule:

$$\text{PR AUC} = \sum_{i=1}^{n-1} (r_{i+1} - r_i) \cdot \frac{p_i + p_{i+1}}{2}$$

where r is recall and p is precision at each threshold.

Value

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

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

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

Multiclass

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

Relevant Level

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

Author(s)

Max Kuhn

See Also

All probability metrics

`pr_curve()` for computing the full precision recall curve.

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

Examples

```
# ------------------------------------------------------------------------------
# Two class example

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

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

# ------------------------------------------------------------------------------
# Multiclass example

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

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

# Change the first level of `obs` from `"VF"` to `"M"` to alter the
```

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

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

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

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

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

`pr_curve` *Precision recall curve*

Description

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

Usage

```
pr_curve(data, ...)

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

```
  event_level = yardstick_event_level(),
  case_weights = NULL
)
```

Arguments

data	A data.frame containing the columns specified by truth and ....
...	A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().

Details

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

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

Value

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

Multiclass

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

Relevant Level

There is no common convention on which factor level should automatically be considered the "event" or "positive" result when computing binary classification metrics. In yardstick, the default is to use the *first* level. To alter this, change the argument event_level to "second" to consider

the *last* level of the factor the level of interest. For multiclass extensions involving one-vs-all comparisons (such as macro averaging), this option is ignored and the "one" level is always the relevant result.

Author(s)

Max Kuhn

See Also

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

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

Examples

```
# ------------------------------------------------------------------------------
# Two class example

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

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


# ------------------------------------------------------------------------------
# `autoplot()`

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

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

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

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

`ranked_prob_score` *Ranked probability scores for ordinal classification models*

Description

Compute the ranked probability score (RPS) for a classification model using ordered classes.

Usage

```
ranked_prob_score(data, ...)

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

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

Arguments

`data`	A `data.frame` containing the columns specified by `truth` and `...`.
`...`	A set of unquoted column names or one or more `dplyr` selector functions to choose which variables contain the class probabilities. If `truth` is binary, only 1 column should be selected, and it should correspond to the value of `event_level`. Otherwise, there should be as many columns as factor levels of `truth` and the ordering of the columns should be the same as the factor levels of `truth`.
`truth`	The column identifier for the true class results (that is an *ordered* `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 with class `ordered`.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`case_weights`	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.
`estimate`	A matrix with as many columns as factor levels of `truth`. *It is assumed that these are in the same order as the levels of* `truth`.

Details

Ranked probability score is a metric that should be minimized. The output ranges from 0 to 1, with 0 indicating perfect predictions.

The ranked probability score is a Brier score for ordinal data that uses the *cumulative* probability of an event (i.e. `Pr[class <= i]` for `i` = 1, 2, ..., `C` classes). These probabilities are compared to indicators for the truth being less than or equal to class `i`.

Since the cumulative sum of a vector of probability predictions add up to one, there is an embedded redundancy in the data. For this reason, the raw mean is divided by the number of classes minus one.

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

Value

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

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

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

Multiclass

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

Author(s)

Max Kuhn

References

Wilks, D. S. (2011). *Statistical Methods in the Atmospheric Sciences*. Academic press. (see Chapter 7)

Janitza, S., Tutz, G., & Boulesteix, A. L. (2016). Random forest for ordinal responses: prediction and variable selection. Computational Statistics and Data Analysis, 96, 57-73. (see Section 2)

Lechner, M., & Okasa, G. (2019). Random forest estimation of the ordered choice model. arXiv preprint arXiv:1907.02436. (see Section 5)

See Also

All ordered probability metrics

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

Examples

```
library(dplyr)
data(hpc_cv)

hpc_cv$obs <- as.ordered(hpc_cv$obs)
```

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

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

`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,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

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

Arguments

data	Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
...	Not currently used.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

Details

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

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

Suppose a 2x2 table with notation:

	Reference	
Predicted	Relevant	Irrelevant
Relevant	A	B
Irrelevant	C	D

The formula used here is:

$$\text{Recall} = \frac{A}{A + C}$$

Recall is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating that all actual positives were predicted as positive.

Value

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

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

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

Relevant Level

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

Multiclass

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

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

All class metrics

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `fall_out()`, `j_index()`, `kap()`, `markedness()`, `mcc()`, `miss_rate()`, `npv()`, `ppv()`, `precision()`, `roc_dist()`, `sedi()`, `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
recall_vec(
  two_class_example$truth,
  two_class_example$predicted,
  event_level = "second"
)
```

---

rmse *Root mean squared error*

Description

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

Usage

```
rmse(data, ...)

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

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

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

Arguments

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

Details

RMSE is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

The formula for RMSE is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{truth}_i - \text{estimate}_i)^2}$$

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

mse() for the mean squared error, which is RMSE without the square root.

All numeric metrics

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

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

---

rmse_relative *Relative root mean squared error*

Description

Calculate the relative root mean squared error. This metric is the root mean squared error normalized by the range of the true values. rmse_relative() is sometimes called normalized RMSE (NRMSE) when range normalization is used.

Usage

```
rmse_relative(data, ...)

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

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

Arguments

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

Details

Relative RMSE is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

The formula for relative RMSE is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\text{truth}_i - \text{estimate}_i)^2}$$

$$\text{Relative RMSE} = \frac{\text{RMSE}}{\max(\text{truth}) - \min(\text{truth})}$$

Note that if all true values are identical (i.e., the range is zero), the result will be Inf.

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

See Also

All numeric metrics

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

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

Examples

```
# Supply truth and predictions as bare column names
rmse_relative(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_relative(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,
  ...,
  estimator = NULL,
  na_rm = TRUE,
  event_level = yardstick_event_level(),
  case_weights = NULL,
  options = list()
)

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

Arguments

data	A `data.frame` containing the columns specified by `truth` and `...`.
...	A set of unquoted column names or one or more `dplyr` selector functions to choose which variables contain the class probabilities. If `truth` is binary, only 1 column should be selected, and it should correspond to the value of `event_level`. Otherwise, there should be as many columns as factor levels of `truth` and the ordering of the columns should be the same as the factor levels of `truth`.
truth	The column identifier for the true class results (that is a `factor`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `factor` vector.
estimator	One of `"binary"`, `"hand_till"`, `"macro"`, or `"macro_weighted"` to specify the type of averaging to be done. `"binary"` is only relevant for the two class case. The others are general methods for calculating multiclass metrics. The default will automatically choose `"binary"` if `truth` is binary, `"hand_till"` if `truth` has >2 levels and `case_weights` isn't specified, or `"macro"` if `truth` has >2 levels and `case_weights` is specified (in which case `"hand_till"` isn't well-defined).
na_rm	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.

event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
options	[deprecated] No longer supported as of yardstick 1.0.0. If you pass something here it will be ignored with a warning. Previously, these were options passed on to pROC::roc(). If you need support for this, use the pROC package directly.
estimate	If truth is binary, a numeric vector of class probabilities corresponding to the "relevant" class. Otherwise, a matrix with as many columns as factor levels of truth. *It is assumed that these are in the same order as the levels of* truth.

Details

ROC AUC is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect discrimination.

The area under the ROC curve is computed using the trapezoidal rule:

$$\text{AUC} = \sum_{i=1}^{n-1} (x_{i+1} - x_i) \cdot \frac{y_i + y_{i+1}}{2}$$

where x is the false positive rate (1 - specificity) and y is the true positive rate (sensitivity) at each threshold.

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

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

Value

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

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

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

Relevant Level

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

Multiclass

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

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

Author(s)

Max Kuhn

References

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

All probability metrics

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

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

Examples

```
# ------------------------------------------------------------------------------
# Two class example

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

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

# ------------------------------------------------------------------------------
# Multiclass example
```

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

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

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

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

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

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

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

`roc_auc_survival` *Time-Dependent ROC AUC for Censored Data*

Description

Compute the area under the ROC survival curve using predicted survival probabilities that corresponds to different time points.

Usage

```
roc_auc_survival(data, ...)

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

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

Arguments

data	A data.frame containing the columns specified by truth and ... .
...	The column identifier for the survival probabilities this should be a list column of data.frames corresponding to the output given when predicting with censored model. 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, the dots are not used.
truth	The column identifier for the true survival result (that is created using survival::Surv().). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, an survival::Surv() object.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
estimate	A list column of data.frames corresponding to the output given when predicting with censored model. See the details for more information regarding format.

Details

ROC AUC survival is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect discrimination.

This formulation takes survival probability predictions at one or more specific *evaluation times* and, for each time, computes the area under the ROC curve. To account for censoring, inverse probability of censoring weights (IPCW) are used in the calculations. See equation 7 of section 4.3 in Blanche *et al* (2013) for the details.

The column passed to ... should be a list column with one element per independent experiential unit (e.g. patient). The list column should contain data frames with several columns:

- .eval_time: The time that the prediction is made.
- .pred_survival: The predicted probability of survival up to .eval_time
- .weight_censored: The case weight for the inverse probability of censoring.

The last column can be produced using parsnip::.censoring_weights_graf(). This corresponds to the weighting scheme of Graf *et al* (1999). The internal data set lung_surv shows an example of the format.

This method automatically groups by the `.eval_time` argument.

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

Value

A `tibble` with columns `.metric`, `.estimator`, and `.estimate`.

For an ungrouped data frame, the result has one row of values. For a grouped data frame, the number of rows returned is the same as the number of groups.

For `roc_auc_survival_vec()`, a `numeric` vector same length as the input argument `eval_time`. (or `NA`).

Author(s)

Emil Hvitfeldt

References

Blanche, P., Dartigues, J.-F. and Jacqmin-Gadda, H. (2013), Review and comparison of ROC curve estimators for a time-dependent outcome with marker-dependent censoring. *Biom. J.*, 55: 687-704.

Graf, E., Schmoor, C., Sauerbrei, W. and Schumacher, M. (1999), Assessment and comparison of prognostic classification schemes for survival data. *Statist. Med.*, 18: 2529-2545.

See Also

All dynamic survival metrics

Compute the ROC survival curve with `roc_curve_survival()`.

Other dynamic survival metrics: `brier_survival()`, `brier_survival_integrated()`

Examples

```
library(dplyr)

lung_surv |>
  roc_auc_survival(
    truth = surv_obj,
    .pred
  )
```

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

Description

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

Usage

```
roc_aunp(data, ...)

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

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

Arguments

`data`	A `data.frame` containing the columns specified by `truth` and `...`.
`...`	A set of unquoted column names or one or more `dplyr` selector functions to choose which variables contain the class probabilities. There should be as many columns as factor levels of `truth`.
`truth`	The column identifier for the true class results (that is a `factor`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `factor` vector.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`case_weights`	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.
`options`	`[deprecated]` No longer supported as of yardstick 1.0.0. If you pass something here it will be ignored with a warning. Previously, these were options passed on to `pROC::roc()`. If you need support for this, use the pROC package directly.

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

Details

ROC AUNP is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect discrimination.

The formula used here is:

$$\text{ROC AUNP} = \sum_{k=1}^{K} p_k \cdot \text{AUC}_k$$

where p_k is the proportion of observations in class k and AUC_k is the binary ROC AUC for class k versus the rest.

Value

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

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

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

Relevant Level

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

Multiclass

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

Author(s)

Julia Silge

References

Ferri, C., Hernández-Orallo, J., & Modroiu, R. (2009). "An experimental comparison of performance measures for classification". *Pattern Recognition Letters*. 30 (1), pp 27-38.

See Also

All probability metrics

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

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

Examples

```
# Multiclass example

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

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

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

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

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

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

roc_aunu *Area under the ROC curve of each class against the rest, using the uniform class distribution*

Description

`roc_aunu()` is a multiclass metric that computes the area under the ROC curve of each class against the rest, using the uniform class distribution. This is equivalent to `roc_auc(estimator = "macro")`.

Usage

```
roc_aunu(data, ...)

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

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

Arguments

`data`	A `data.frame` containing the columns specified by `truth` and `...`.
`...`	A set of unquoted column names or one or more `dplyr` selector functions to choose which variables contain the class probabilities. There should be as many columns as factor levels of `truth`.
`truth`	The column identifier for the true class results (that is a `factor`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `factor` vector.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`case_weights`	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.
`options`	`[deprecated]` No longer supported as of yardstick 1.0.0. If you pass something here it will be ignored with a warning. Previously, these were options passed on to `pROC::roc()`. If you need support for this, use the pROC package directly.

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

Details

ROC AUNU is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect discrimination.

The formula used here is:

$$\text{ROC AUNU} = \frac{1}{K}\sum_{k=1}^{K}\text{AUC}_k$$

where K is the number of classes and AUC_k is the binary ROC AUC for class k versus the rest.

Value

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

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

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

Relevant Level

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

Multiclass

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

Author(s)

Julia Silge

References

Ferri, C., Hernández-Orallo, J., & Modroiu, R. (2009). "An experimental comparison of performance measures for classification". *Pattern Recognition Letters*. 30 (1), pp 27-38.

See Also

All probability metrics

`roc_aunp()` for computing the area under the ROC curve of each class against the rest, using the a priori class distribution.

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

Examples

```
# Multiclass example

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

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

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

# Groups are respected
hpc_cv |>
  group_by(Resample) |>
  roc_aunu(obs, VF:L)

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

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

roc_curve *Receiver operator curve*

Description

roc_curve() constructs the full ROC curve and returns a tibble. See roc_auc() for the area under the ROC curve.

Usage

```
roc_curve(data, ...)

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

Arguments

data	A data.frame containing the columns specified by truth and ....
...	A set of unquoted column names or one or more dplyr selector functions to choose which variables contain the class probabilities. If truth is binary, only 1 column should be selected, and it should correspond to the value of event_level. Otherwise, there should be as many columns as factor levels of truth and the ordering of the columns should be the same as the factor levels of truth.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().

`options` [deprecated]

No longer supported as of yardstick 1.0.0. If you pass something here it will be ignored with a warning.

Previously, these were options passed on to `pROC::roc()`. If you need support for this, use the pROC package directly.

`thresholds` A numeric vector to denote what values of `estimate` we calculate the curve for. Defaults to `NULL` which denotes that all unique values of `estimate` are used. Duplicate values are silently removed.

Details

`roc_curve()` computes the sensitivity at every unique value of the probability column (in addition to infinity and minus infinity).

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

Value

A tibble with class `roc_df` or `roc_grouped_df` having columns `.threshold`, `specificity`, and `sensitivity`.

Multiclass

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

Relevant Level

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

Author(s)

Max Kuhn

See Also

Compute the area under the ROC curve with `roc_auc()`.

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

Examples

```
# ------------------------------------------------------------------------------
# Two class example

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

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


# ------------------------------------------------------------------------------
# `autoplot()`

# Visualize the curve using ggplot2 manually
library(ggplot2)
library(dplyr)
roc_curve(two_class_example, truth, Class1) |>
  ggplot(aes(x = 1 - specificity, y = sensitivity)) +
  geom_path() +
  geom_abline(lty = 3) +
  coord_equal() +
  theme_bw()

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

## Not run:

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

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

## End(Not run)
```

`roc_curve_survival` *Time-Dependent ROC surve for Censored Data*

Description

Compute the ROC survival curve using predicted survival probabilities that corresponds to different time points.

Usage

```
roc_curve_survival(data, ...)

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

Arguments

`data`	A `data.frame` containing the columns specified by `truth` and `...`.
`...`	The column identifier for the survival probabilities this should be a list column of data.frames corresponding to the output given when predicting with censored model. 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, the dots are not used.
`truth`	The column identifier for the true survival result (that is created using `survival::Surv()`.). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, an `survival::Surv()` object.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`case_weights`	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.

Details

This formulation takes survival probability predictions at one or more specific *evaluation times* and, for each time, computes the ROC curve. To account for censoring, inverse probability of censoring weights (IPCW) are used in the calculations. See equation 7 of section 4.3 in Blanche *et al* (2013) for the details.

The column passed to `...` should be a list column with one element per independent experiential unit (e.g. patient). The list column should contain data frames with several columns:

- `.eval_time`: The time that the prediction is made.
- `.pred_survival`: The predicted probability of survival up to `.eval_time`
- `.weight_censored`: The case weight for the inverse probability of censoring.

The last column can be produced using `parsnip::.censoring_weights_graf()`. This corresponds to the weighting scheme of Graf *et al* (1999). The internal data set `lung_surv` shows an example of the format.

This method automatically groups by the `.eval_time` argument.

Value

A tibble with class `roc_survival_df`, `grouped_roc_survival_df` having columns `.threshold`, `sensitivity`, `specificity`, and `.eval_time`.

Author(s)

Emil Hvitfeldt

References

Blanche, P., Dartigues, J.-F. and Jacqmin-Gadda, H. (2013), Review and comparison of ROC curve estimators for a time-dependent outcome with marker-dependent censoring. *Biom. J.*, 55: 687-704.

Graf, E., Schmoor, C., Sauerbrei, W. and Schumacher, M. (1999), Assessment and comparison of prognostic classification schemes for survival data. *Statist. Med.*, 18: 2529-2545.

See Also

Compute the area under the ROC survival curve with `roc_auc_survival()`.

Examples

```
result <- roc_curve_survival(
  lung_surv,
  truth = surv_obj,
  .pred
)
result

# ------------------------------------------------------------------------------
# `autoplot()`

# Visualize the curve using ggplot2 manually
library(ggplot2)
library(dplyr)
result |>
  mutate(.eval_time = format(.eval_time)) |>
  ggplot(aes(
    x = 1 - specificity, y = sensitivity,
    group = .eval_time, col = .eval_time
  )) +
  geom_step(direction = "hv") +
  geom_abline(lty = 3) +
  coord_equal() +
  theme_bw()
```

```
# Or use autoplot
autoplot(result)
```

`roc_dist` *Distance to ROC corner*

Description

`roc_dist()` calculates the Euclidean distance from the observed (sensitivity, specificity) point to the ideal corner (1, 1) in ROC space. This is equivalent to the distance from (FPR, TPR) to (0, 1).

This metric is sometimes called "closest to top-left" in ROC analysis and provides an alternative to `j_index()` for finding optimal classification thresholds.

Usage

```
roc_dist(data, ...)

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

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

Arguments

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

truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

Details

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

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

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

$$\text{roc\_dist} = \sqrt{(1-\text{Sensitivity})^2 + (1-\text{Specificity})^2}$$

roc_dist is a metric that should be minimized. The output ranges from 0 to 1.4142135623731, with 0 indicating perfect sensitivity and specificity.

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

Relevant Level

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

Multiclass

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

See Also

All class metrics

`j_index()` for Youden's J statistic, another metric for measuring closeness to the ideal classification point.

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

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv |>
  filter(Resample == "Fold01") |>
  roc_dist(obs, pred)

# Groups are respected
hpc_cv |>
  group_by(Resample) |>
  roc_dist(obs, pred)

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

# Vector version
roc_dist_vec(
  two_class_example$truth,
```

```
  two_class_example$predicted
)

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

`royston_survival` *Royston-Sauerbei D statistic*

Description

Compute the Royston-Sauerbei D statistic

Usage

```
royston_survival(data, ...)

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

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

Arguments

`data`	A `data.frame` containing the columns specified by `truth` and `....`
`...`	Not currently used.
`truth`	The column identifier for the true survival result (that is created using `survival::Surv()`.). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, an `survival::Surv()` object.
`estimate`	The column identifier for the predicted linear predictor, this should be a numeric variable. 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.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.
`case_weights`	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in `data`. For `_vec()` functions, a numeric vector, `hardhat::importance_weights()`, or `hardhat::frequency_weights()`.

Details

Royston D statistic is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect prognostic separation.

Royston and Sauerbrei proposed R^2_D as a measure of explained variation on the log relative hazard scale based on the authors' D statistic. D measures prognostic separation of survival curves, and is closely related to the standard deviation of the prognostic index.

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

Value

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

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

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

Author(s)

Hannah Frick

References

Royston, P., Sauerbrei, W., "A new measure of prognostic separation in survival data", Statistics in Medicine, 23, 723-748, 2004.

See Also

All linear predictor survival metrics

Examples

```
royston_survival(
  data = lung_surv,
  truth = surv_obj,
  estimate = .pred_linear_pred
)
```

---

rpd *Ratio of performance to deviation*

Description

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

Usage

```
rpd(data, ...)

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

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

Arguments

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

Details

In the field of spectroscopy in particular, the ratio of performance to deviation (RPD) has been used as the standard way to report the quality of a model. It is the ratio between the standard deviation of a variable and the standard error of prediction of that variable by a given model. However, its systematic use has been criticized by several authors, since using the standard deviation to represent the spread of a variable can be misleading on skewed dataset. The ratio of performance to interquartile has been introduced by Bellon-Maurel et al. (2010) to address some of these issues, and generalise the RPD to non-normally distributed variables.

RPD is a metric that should be maximized. The output ranges from 0 to Inf, with higher values indicating better model performance.

The formula for RPD is:

$$\text{RPD} = \frac{\text{sd}(\text{truth})}{\text{RMSE}}$$

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

All numeric metrics

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

Other numeric metrics: `ccc()`, `gini_coef()`, `huber_loss()`, `huber_loss_pseudo()`, `iic()`, `mae()`, `mape()`, `mase()`, `mpe()`, `msd()`, `mse()`, `poisson_log_loss()`, `rmse()`, `rmse_relative()`, `rpiq()`, `rsq()`, `rsq_trad()`, `smape()`

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

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, case_weights = NULL, ...)

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

Arguments

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

Details

RPIQ is a metric that should be maximized. The output ranges from 0 to Inf, with higher values indicating better model performance.

The formula for RPIQ is:

$$\text{RPIQ} = \frac{\text{IQR}(\text{truth})}{\text{RMSE}}$$

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

All numeric metrics

The closely related deviation metric: rpd()

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

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

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, case_weights = NULL, ...)

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

Arguments

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

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`.

Because `rsq()` internally computes a correlation, if either `truth` or `estimate` are constant it can result in a divide by zero error. In these cases, a warning is thrown and `NA` is returned. This can occur when a model predicts a single value for all samples. For example, a regularized model that eliminates all predictors except for the intercept would do this. Another example would be a CART model that contains no splits.

R squared is a metric that should be maximized. The output ranges from -Inf to 1, with 1 indicating perfect predictions.

The formula for R squared is:

$$\text{rsq} = \frac{\text{cov}(\text{truth}, \text{estimate})^2}{\text{var}(\text{truth}) \cdot \text{var}(\text{estimate})}$$

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

All numeric metrics

Other numeric metrics: `ccc()`, `gini_coef()`, `huber_loss()`, `huber_loss_pseudo()`, `iic()`, `mae()`, `mape()`, `mase()`, `mpe()`, `msd()`, `mse()`, `poisson_log_loss()`, `rmse()`, `rmse_relative()`, `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)

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

`rsq_trad` *R squared - traditional*

Description

Calculate the coefficient of determination using the traditional definition of R squared using sum of squares. For a measure of R squared that is strictly between (0, 1), see `rsq()`.

Usage

```
rsq_trad(data, ...)
```

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

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

Arguments

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

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.

Traditional R squared is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating perfect predictions. Negative values can occur when the model is non-informative.

The formula for traditional R squared is:

$$\text{rsq\_trad} = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{n}(\text{truth}_i - \text{estimate}_i)^2}{\sum_{i=1}^{n}(\text{truth}_i - \bar{\text{truth}})^2}$$

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

All numeric metrics

Other numeric metrics: ccc(), gini_coef(), huber_loss(), huber_loss_pseudo(), iic(), mae(), mape(), mase(), mpe(), msd(), mse(), poisson_log_loss(), rmse(), rmse_relative(), 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)
```

sedi *Symmetric Extremal Dependence Index*

Description

Symmetric Extremal Dependence Index (SEDI) is a skill metric for classification that remains reliable at extreme prevalence levels where traditional metrics (TSS, MCC, Kappa) degrade. It is defined using the hit rate (sensitivity) and false alarm rate (1 - specificity):

$$\text{SEDI} = \frac{\ln F - \ln H - \ln(1-F) + \ln(1-H)}{\ln F + \ln H + \ln(1-F) + \ln(1-H)}$$

where H is sensitivity (hit rate) and F is the false alarm rate (1 - specificity).

Usage

```
sedi(data, ...)

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

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

Arguments

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

truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

Details

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formulas used here are:

$$H = \text{Sensitivity} = \frac{A}{A + C}$$

$$F = 1 - \text{Specificity} = \frac{B}{B + D}$$

SEDI is a metric that should be maximized. The output ranges from -1 to 1, with 1 indicating perfect discrimination.

SEDI is **base-rate independent**: its value depends only on sensitivity and specificity (class-conditional rates), not on prevalence. The logarithmic transformation ensures the metric remains discriminating even when events are extremely rare (prevalence < 2.5%), where j_index() (TSS) converges to the hit rate alone and mcc() exhibits denominator suppression.

When sensitivity or specificity is exactly 0 or 1, the logarithm is undefined. A small constant (1e-9) is used to clamp values away from these boundaries.

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 `sedi_vec()`, a single `numeric` value (or `NA`).

Prevalence guidance

- **Prevalence >= 10%**: MCC, TSS, and SEDI all perform well.
- **Prevalence 2.5-10%**: SEDI preferred; MCC and TSS still usable.
- **Prevalence < 2.5%**: SEDI strongly recommended; MCC and TSS unreliable.

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.

For multiclass problems, SEDI is computed via one-vs-all decomposition: each class is treated as a binary problem against all other classes, and a per-class SEDI is calculated. Macro averaging (the default) weights all classes equally, which is recommended since SEDI's log transform already handles class imbalance internally. Macro-weighted averaging weights by class prevalence. Micro averaging pools counts across classes before computing a single SEDI value.

Relevant Level

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

Author(s)

Simon Dedman

References

Ferro, C.A.T. and Stephenson, D.B. (2011). "Extremal Dependence Indices: Improved Verification Measures for Deterministic Forecasts of Rare Binary Events". Weather and Forecasting. 26 (5): 699-713.

Wunderlich, R.F., Lin, Y.-P., Anthony, J. and Petway, J.R. (2019). "Two alternative evaluation metrics to replace the true skill statistic in the assessment of species distribution models". Nature Conservation. 35: 97-116.

See Also

All class metrics

Other class metrics: accuracy(), bal_accuracy(), detection_prevalence(), f_meas(), fall_out(), j_index(), kap(), markedness(), mcc(), miss_rate(), npv(), ppv(), precision(), recall(), roc_dist(), sens(), spec()

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv |>
  filter(Resample == "Fold01") |>
  sedi(obs, pred)

# Groups are respected
hpc_cv |>
  group_by(Resample) |>
  sedi(obs, pred)

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

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

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

---

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,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

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

sensitivity(data, ...)

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

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

Arguments

data	Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
...	Not currently used.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

Details

The sensitivity (sens()) is defined as the proportion of positive results out of the number of samples which were actually positive. For positive observations, the proportion of model predictions that correctly predicted positive.

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

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formula used here is:

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

Sensitivity is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating that all actual positives were predicted as positive.

Value

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

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

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

Relevant Level

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

Multiclass

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

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

All class metrics

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `fall_out()`, `j_index()`, `kap()`, `markedness()`, `mcc()`, `miss_rate()`, `npv()`, `ppv()`, `precision()`, `recall()`, `roc_dist()`, `sedi()`, `spec()`

Other sensitivity metrics: `fall_out()`, `miss_rate()`, `npv()`, `ppv()`, `spec()`

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

hpc_cv |>
  filter(Resample == "Fold01") |>
  sens(obs, pred)

# Groups are respected
hpc_cv |>
  group_by(Resample) |>
  sens(obs, pred)

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

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

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

---

smape *Symmetric mean absolute percentage error*

Description

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

Usage

```
smape(data, ...)

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

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

Arguments

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

Details

This implementation of smape() is the "usual definition" where the denominator is divided by two.

SMAPE is a metric that should be minimized. The output ranges from 0 to 100, with 0 indicating perfect predictions.

The formula for SMAPE is:

$$\text{SMAPE} = \frac{100}{n} \sum_{i=1}^{n} \frac{|\text{estimate}_i - \text{truth}_i|}{(|\text{truth}_i| + |\text{estimate}_i|)/2}$$

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

All numeric metrics

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

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

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,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

spec_vec(
  truth,
  estimate,
  estimator = NULL,
```

```
  na_rm = TRUE,
  case_weights = NULL,
  event_level = yardstick_event_level(),
  ...
)

specificity(data, ...)

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

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

Arguments

data	Either a data.frame containing the columns specified by the truth and estimate arguments, or a table/matrix where the true class results should be in the columns of the table.
...	Not currently used.
truth	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a factor vector.
estimate	The column identifier for the predicted class results (that is also factor). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a factor vector.
estimator	One of: "binary", "macro", "macro_weighted", or "micro" to specify the type of averaging to be done. "binary" is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose "binary" or "macro" based on estimate.

na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().
event_level	A single string. Either "first" or "second" to specify which level of truth to consider as the "event". This argument is only applicable when estimator = "binary". The default uses an internal helper that defaults to "first".

Details

The specificity measures the proportion of negatives that are correctly identified as negatives. For negative observations, the proportion of model predictions that correctly predicted negative.

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

Suppose a 2x2 table with notation:

	Reference	
Predicted	Positive	Negative
Positive	A	B
Negative	C	D

The formula used here is:

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

Specificity is a metric that should be maximized. The output ranges from 0 to 1, with 1 indicating that all actual negatives were predicted as 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 spec_vec(), a single numeric value (or NA).

Relevant Level

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

Multiclass

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

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

All class metrics

Other class metrics: `accuracy()`, `bal_accuracy()`, `detection_prevalence()`, `f_meas()`, `fall_out()`, `j_index()`, `kap()`, `markedness()`, `mcc()`, `miss_rate()`, `npv()`, `ppv()`, `precision()`, `recall()`, `roc_dist()`, `sedi()`, `sens()`

Other sensitivity metrics: `fall_out()`, `miss_rate()`, `npv()`, `ppv()`, `sens()`

Examples

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

# Multiclass
library(dplyr)
data(hpc_cv)

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

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

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

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

```
)

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

`summary.conf_mat` *Summary Statistics for Confusion Matrices*

Description

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

Usage

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

Arguments

`object`	An object of class `conf_mat()`.
`prevalence`	A number in `(0, 1)` for the prevalence (i.e. prior) of the event. If left to the default, the data are used to derive this value.
`beta`	A numeric value used to weight precision and recall for `f_meas()`.
`estimator`	One of: `"binary"`, `"macro"`, `"macro_weighted"`, or `"micro"` to specify the type of averaging to be done. `"binary"` is only relevant for the two class case. The other three are general methods for calculating multiclass metrics. The default will automatically choose `"binary"` or `"macro"` based on `estimate`.
`event_level`	A single string. Either `"first"` or `"second"` to specify which level of `truth` to consider as the "event". This argument is only applicable when `estimator = "binary"`. The default uses an internal helper that defaults to `"first"`.
`...`	Not currently used.

Value

A tibble containing various classification metrics.

Relevant Level

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

See Also

`conf_mat()`

Examples

```
data("two_class_example")

cmat <- conf_mat(two_class_example, truth = "truth", estimate = "predicted")
summary(cmat)
summary(cmat, prevalence = 0.70)

library(dplyr)
library(tidyr)
data("hpc_cv")

# Compute statistics per resample then summarize
all_metrics <- hpc_cv |>
  group_by(Resample) |>
  conf_mat(obs, pred) |>
  mutate(summary_tbl = lapply(conf_mat, summary)) |>
  unnest(summary_tbl)

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

`two_class_example` *Two Class Predictions*

Description

Two Class Predictions

Details

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

Value

`two_class_example`
: a data frame

Examples

```
data(two_class_example)
str(two_class_example)

# `truth` is a 2 level factor. The first level is `"Class1"`, which is the
# "event of interest" by default in yardstick. See the Relevant Level
# section in any classification function (such as `?pr_auc`) to see how
# to change this.
levels(hpc_cv$obs)
```

`weighted_interval_score` *Compute weighted interval score*

Description

Weighted interval score (WIS), a well-known quantile-based approximation of the commonly-used continuous ranked probability score (CRPS). WIS is a proper score, and can be thought of as a distributional generalization of absolute error. For example, see Bracher et al. (2020) for discussion in the context of COVID-19 forecasting.

Usage

```
weighted_interval_score(data, ...)

## S3 method for class 'data.frame'
weighted_interval_score(
  data,
  truth,
  estimate,
  quantile_levels = NULL,
  na_rm = TRUE,
  quantile_estimate_nas = c("impute", "drop", "propagate"),
  case_weights = NULL,
  ...
)

weighted_interval_score_vec(
  truth,
  estimate,
  quantile_levels = NULL,
  na_rm = FALSE,
```

```
  quantile_estimate_nas = c("impute", "drop", "propagate"),
  case_weights = NULL,
  ...
)
```

Arguments

data	A data.frame containing the columns specified by the truth and estimate arguments.
...	Not Currently used.
truth	The column identifier for the true class results (that is a 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 class results (that is also quantile_pred). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a quantile_pred vector.
quantile_levels	probabilities. If specified, the score will be computed at this set of levels. Otherwise, those present in x will be used. If quantile_levels do not exactly match those available in x, then some quantiles will have implicit missingness. Handling of these is determined by quantile_estimate_nas.
na_rm	logical. If TRUE, missing values in actual or both implicit and explicit (values of NA present in x), will be ignored (dropped) in the calculation of the summary score. If FALSE (the default), any NAs will result in the summary being NA.
quantile_estimate_nas	character. This argument applies only to x. It handles imputation of individual quantile_levels that are necessary to compute a score. Because each element of x is a hardhat::quantile_pred, it is possible for these to be missing for particular quantile_levels. There are a number of different possibilities for such missingness. The options are as follows: • For "impute", both explicit and implicit missing values will be imputed using hardhat::impute_quantiles() prior to the calculation of the score. So the score will be NA only if imputation fails. • For "drop", any explicit missing values will be removed before calculating the score for a particular prediction. This may be reasonable due to the weighting. For example, if the estimate has quantile_levels = c(.25, .5, .75) but the median is NA for a particular prediction, it may be reasonable to average the accuracy of c(.25, .75) for that prediction with others that don't have missingness. This option is only works if quantile_levels = NULL or is a subset of the quantile_levels in x. • For "propagate", any missing value predictions will result in that element of x having a score of NA. If na_rm = TRUE, then these will be removed before averaging.
case_weights	The optional column identifier for case weights. This should be an unquoted column name that evaluates to a numeric column in data. For _vec() functions, a numeric vector, hardhat::importance_weights(), or hardhat::frequency_weights().

Details

Weighted interval score is a metric that should be minimized. The output ranges from 0 to Inf, with 0 indicating perfect predictions.

Value

a vector of nonnegative scores.

See Also

All quantile metrics

Examples

```
library(hardhat)

quantile_levels <- c(.2, .4, .6, .8)
pred1 <- 1:4
pred2 <- 8:11
preds <- quantile_pred(rbind(pred1, pred2), quantile_levels)
truth <- c(3.3, 7.1)
weighted_interval_score_vec(truth, preds)
weighted_interval_score_vec(truth, preds, quantile_levels = c(.25, .5, .75))

# Missing value behaviours

preds_na <- quantile_pred(rbind(pred1, c(1, 2, NA, 4)), 1:4 / 5)
truth <- c(2.5, 2.5)
weighted_interval_score_vec(truth, preds_na)
weighted_interval_score_vec(truth, preds_na, quantile_levels = 1:9 / 10)
try(weighted_interval_score_vec(
  truth,
  preds_na,
  quantile_levels = 1:9 / 10,
  quantile_estimate_nas = "drop"
))
weighted_interval_score_vec(
  truth,
  preds_na,
  quantile_levels = c(2, 3) / 5,
  quantile_estimate_nas = "drop"
)
weighted_interval_score_vec(
  truth, preds_na, na_rm = TRUE, quantile_estimate_nas = "propagate"
)
weighted_interval_score_vec(
  truth, preds_na, quantile_estimate_nas = "propagate"
)
```

yardstick_remove_missing *Developer function for handling missing values in new metrics*

Description

yardstick_remove_missing(), and yardstick_any_missing() are useful alongside the metric-summarizers functions for implementing new custom metrics. yardstick_remove_missing() removes any observations that contains missing values across, truth, estimate and case_weights. yardstick_any_missing() returns FALSE if there is any missing values in the inputs.

Usage

```
yardstick_remove_missing(truth, estimate, case_weights)

yardstick_any_missing(truth, estimate, case_weights)
```

Arguments

truth, estimate	Vectors of the same length.
case_weights	A vector of the same length as truth and estimate, or NULL if case weights are not being used.

See Also

metric-summarizers

Index