# Package: probably (via r-universe)

September 26, 2024

**Title** Tools for Post-Processing Predicted Values

**Version** 1.0.3.9001

**Description** Models can be improved by post-processing class probabilities, by: recalibration, conversion to hard probabilities, assessment of equivocal zones, and other activities. 'probably' contains tools for conducting these operations as well as calibration tools and conformal inference techniques for regression models.

**License** MIT + file LICENSE

**URL** https://github.com/tidymodels/probably, https://probably.tidymodels.org

**BugReports** https://github.com/tidymodels/probably/issues

**Depends** R (>= 3.6)

**Imports** butcher, cli, dplyr (>= 1.1.0), furrr, generics (>= 0.1.3), ggplot2, hardhat, pillar, purrr, rlang (>= 1.1.0), tidyr (>= 1.3.0), tidyselect (>= 1.1.2), tune (>= 1.1.2), vctrs (>= 0.4.1), withr, workflows (>= 1.1.4), yardstick (>= 1.3.0)

**Suggests** betacal, covr, knitr, MASS, mgcv, modeldata (>= 1.1.0), nnet, parsnip (>= 1.2.0), quantregForest, randomForest, recipes, rmarkdown, rsample, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**ByteCompile** true

**Config/Needs/website** tidyverse/tidytemplate

**Config/testthat/edition** 3

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**Repository** https://tidymodels.r-universe.dev

**RemoteUrl** https://github.com/tidymodels/probably

**RemoteRef** HEAD

**RemoteSha** 545f9ab7edbc22eb005a48525fcb7d7e1cf21694

# Contents

append_class_pred *Add a* class_pred *column*

## Description

This function is similar to make_class_pred(), but is useful when you have a large number of class probability columns and want to use tidyselect helpers. It appends the new class_pred vector as a column on the original data frame.

## Usage

```
append_class_pred(
  .data,
  ...,
  levels,
  ordered = FALSE,
  min_prob = 1/length(levels),
  name = ".class_pred"
)
```

## Arguments

| | |
|---|---|
| .data | A data frame or tibble. |
| ... | One or more unquoted expressions separated by commas to capture the columns of .data containing the class probabilities. You can treat variable names like they are positions, so you can use expressions like x:y to select ranges of variables or use selector functions to choose which columns. For make_class_pred, the columns for all class probabilities should be selected (in the same order as the levels object). For two_class_pred, a vector of class probabilities should be selected. |
| levels | A character vector of class levels. The length should be the same as the number of selections made through ..., or length 2 for make_two_class_pred(). |
| ordered | A single logical to determine if the levels should be regarded as ordered (in the order given). This results in a class_pred object that is flagged as ordered. |
| min_prob | A single numeric value. If any probabilities are less than this value (by row), the row is marked as *equivocal*. |
| name | A single character value for the name of the appended class_pred column. |

## Value

.data with an extra class_pred column appended onto it.

## Examples

```
# The following two examples are equivalent and demonstrate
# the helper, append_class_pred()

library(dplyr)

species_probs %>%
  mutate(
    .class_pred = make_class_pred(
      .pred_bobcat, .pred_coyote, .pred_gray_fox,
      levels = levels(Species),
      min_prob = .5
    )
  )

lvls <- levels(species_probs$Species)

append_class_pred(
  .data = species_probs,
  contains(".pred_"),
  levels = lvls,
  min_prob = .5
)
```

---

# as_class_pred *Coerce to a* class_pred *object*

---

## Description

as_class_pred() provides coercion to class_pred from other existing objects.

## Usage

```
as_class_pred(x, which = integer(), equivocal = "[EQ]")
```

## Arguments

| | |
|---|---|
| x | A factor or ordered factor. |
| which | An integer vector specifying the locations of x to declare as equivocal. |
| equivocal | A single character specifying the equivocal label used when printing. |

## Examples

```
x <- factor(c("Yes", "No", "Yes", "Yes"))
as_class_pred(x)
```

## boosting_predictions *Boosted regression trees predictions*

### Description

Boosted regression trees predictions

### Details

These data have a set of holdout predictions from 10-fold cross-validation and a separate collection of test set predictions from the same boosted tree model. The data were generated using the `sim_regression` function in the **modeldata** package.

### Value

`boosting_predictions_oob`, `boosting_predictions_test`
: tibbles

### Examples

```
data(boosting_predictions_oob)
str(boosting_predictions_oob)
str(boosting_predictions_test)
```

## bound_prediction *Truncate a numeric prediction column*

### Description

For user-defined `lower_limit` and/or `upper_limit` bound, ensure that the values in the `.pred` column are coerced to these bounds.

### Usage

```
bound_prediction(
  x,
  lower_limit = -Inf,
  upper_limit = Inf,
  call = rlang::current_env()
)
```

### Arguments

| | |
|---|---|
| `x` | A data frame that contains a numeric column named `.pred`. |
| `lower_limit`, `upper_limit` | Single numerics (or `NA`) that define constraints on `.pred`. |
| `call` | The call to be displayed in warnings or errors. |

## Value

x with potentially adjusted values.

## Examples

```
data(solubility_test, package = "yardstick")

names(solubility_test) <- c("solubility", ".pred")

bound_prediction(solubility_test, lower_limit = -1)
```

---

# cal_apply *Applies a calibration to a set of existing predictions*

---

## Description

Applies a calibration to a set of existing predictions

## Usage

```
cal_apply(.data, object, pred_class = NULL, parameters = NULL, ...)

## S3 method for class 'data.frame'
cal_apply(.data, object, pred_class = NULL, parameters = NULL, ...)

## S3 method for class 'tune_results'
cal_apply(.data, object, pred_class = NULL, parameters = NULL, ...)

## S3 method for class 'cal_object'
cal_apply(.data, object, pred_class = NULL, parameters = NULL, ...)
```

## Arguments

| | |
|---|---|
| .data | An object that can process a calibration object. |
| object | The calibration object (cal_object). |
| pred_class | (Optional, classification only) Column identifier for the hard class predictions (a factor vector). This column will be adjusted based on changes to the calibrated probability columns. |
| parameters | (Optional) An optional tibble of tuning parameter values that can be used to filter the predicted values before processing. Applies only to tune_results objects. |
| ... | Optional arguments; currently unused. |

## Details

cal_apply() currently supports data.frames only. It extracts the truth and the estimate columns names from the calibration object.

## See Also

https://www.tidymodels.org/learn/models/calibration/, cal_estimate_beta(), cal_estimate_isotonic(), cal_estimate_isotonic_boot(), cal_estimate_linear(), cal_estimate_logistic(), cal_estimate_multinomial(

## Examples

```
# ------------------------------------------------------------------------------
# classification example

w_calibration <- cal_estimate_logistic(segment_logistic, Class)

cal_apply(segment_logistic, w_calibration)
```

---

# cal_estimate_beta *Uses a Beta calibration model to calculate new probabilities*

---

## Description

Uses a Beta calibration model to calculate new probabilities

## Usage

```
cal_estimate_beta(
  .data,
  truth = NULL,
  shape_params = 2,
  location_params = 1,
  estimate = dplyr::starts_with(".pred_"),
  parameters = NULL,
  ...
)

## S3 method for class 'data.frame'
cal_estimate_beta(
  .data,
  truth = NULL,
  shape_params = 2,
  location_params = 1,
  estimate = dplyr::starts_with(".pred_"),
  parameters = NULL,
  ...,
  .by = NULL
)

## S3 method for class 'tune_results'
cal_estimate_beta(
  .data,
```

```
  truth = NULL,
  shape_params = 2,
  location_params = 1,
  estimate = dplyr::starts_with(".pred_"),
  parameters = NULL,
  ...
)

## S3 method for class 'grouped_df'
cal_estimate_beta(
  .data,
  truth = NULL,
  shape_params = 2,
  location_params = 1,
  estimate = NULL,
  parameters = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `.data` | An ungrouped `data.frame` object, or `tune_results` object, that contains predictions and probability columns. |
| `truth` | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| `shape_params` | Number of shape parameters to use. Accepted values are 1 and 2. Defaults to 2. |
| `location_params` | Number of location parameters to use. Accepted values 1 and 0. Defaults to 1. |
| `estimate` | A vector of column identifiers, or one of `dplyr` selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (`.pred_`). The order of the identifiers will be considered the same as the order of the levels of the `truth` variable. |
| `parameters` | (Optional) An optional tibble of tuning parameter values that can be used to filter the predicted values before processing. Applies only to `tune_results` objects. |
| `...` | Additional arguments passed to the models or routines used to calculate the new probabilities. |
| `.by` | The column identifier for the grouping variable. This should be a single unquoted column name that selects a qualitative variable for grouping. Default to `NULL`. When `.by = NULL` no grouping will take place. |

## Details

This function uses the `betacal::beta_calibration()` function, and retains the resulting model.

## Multiclass Extension

This method is designed to work with two classes. For multiclass, it creates a set of "one versus all" calibrations for each class. After they are applied to the data, the probability estimates are

re-normalized to add to one. This final step might compromise the calibration.

## References

Meelis Kull, Telmo M. Silva Filho, Peter Flach "Beyond sigmoids: How to obtain well-calibrated probabilities from binary classifiers with beta calibration," *Electronic Journal of Statistics* 11(2), 5052-5080, (2017)

## See Also

https://www.tidymodels.org/learn/models/calibration/, cal_validate_beta()

## Examples

```
# It will automatically identify the probability columns
# if passed a model fitted with tidymodels
cal_estimate_beta(segment_logistic, Class)
```

---

# cal_estimate_isotonic *Uses an Isotonic regression model to calibrate model predictions.*

---

## Description

Uses an Isotonic regression model to calibrate model predictions.

## Usage

```
cal_estimate_isotonic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  parameters = NULL,
  ...
)

## S3 method for class 'data.frame'
cal_estimate_isotonic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  parameters = NULL,
  ...,
  .by = NULL
)

## S3 method for class 'tune_results'
cal_estimate_isotonic(
  .data,
```

```
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  parameters = NULL,
  ...
)

## S3 method for class 'grouped_df'
cal_estimate_isotonic(
  .data,
  truth = NULL,
  estimate = NULL,
  parameters = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| .data | An ungrouped data.frame object, or tune_results object, that contains predictions and probability columns. |
| truth | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| estimate | A vector of column identifiers, or one of dplyr selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (.pred_). The order of the identifiers will be considered the same as the order of the levels of the truth variable. |
| parameters | (Optional) An optional tibble of tuning parameter values that can be used to filter the predicted values before processing. Applies only to tune_results objects. |
| ... | Additional arguments passed to the models or routines used to calculate the new probabilities. |
| .by | The column identifier for the grouping variable. This should be a single unquoted column name that selects a qualitative variable for grouping. Default to NULL. When .by = NULL no grouping will take place. |

## Details

This function uses stats::isoreg() to create obtain the calibration values for binary classification or numeric regression.

## Multiclass Extension

This method is designed to work with two classes. For multiclass, it creates a set of "one versus all" calibrations for each class. After they are applied to the data, the probability estimates are re-normalized to add to one. This final step might compromise the calibration.

## References

Zadrozny, Bianca and Elkan, Charles. (2002). Transforming Classifier Scores into Accurate Multiclass Probability Estimates. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

## See Also

https://www.tidymodels.org/learn/models/calibration/, cal_validate_isotonic()

## Examples

```
# ------------------------------------------------------------------------------
# Binary Classification

# It will automatically identify the probability columns
# if passed a model fitted with tidymodels
cal_estimate_isotonic(segment_logistic, Class)

# Specify the variable names in a vector of unquoted names
cal_estimate_isotonic(segment_logistic, Class, c(.pred_poor, .pred_good))

# dplyr selector functions are also supported
cal_estimate_isotonic(segment_logistic, Class, dplyr::starts_with(".pred_"))

# ------------------------------------------------------------------------------
# Regression (numeric outcomes)

cal_estimate_isotonic(boosting_predictions_oob, outcome, .pred)
```

---

# cal_estimate_isotonic_boot

*Uses a bootstrapped Isotonic regression model to calibrate probabilities*

---

## Description

Uses a bootstrapped Isotonic regression model to calibrate probabilities

## Usage

```
cal_estimate_isotonic_boot(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  times = 10,
  parameters = NULL,
  ...
)

## S3 method for class 'data.frame'
cal_estimate_isotonic_boot(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
```

```
  times = 10,
  parameters = NULL,
  ...,
  .by = NULL
)

## S3 method for class 'tune_results'
cal_estimate_isotonic_boot(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  times = 10,
  parameters = NULL,
  ...
)

## S3 method for class 'grouped_df'
cal_estimate_isotonic_boot(
  .data,
  truth = NULL,
  estimate = NULL,
  times = 10,
  parameters = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `.data` | An ungrouped `data.frame` object, or `tune_results` object, that contains predictions and probability columns. |
| `truth` | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| `estimate` | A vector of column identifiers, or one of `dplyr` selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (`.pred_`). The order of the identifiers will be considered the same as the order of the levels of the `truth` variable. |
| `times` | Number of bootstraps. |
| `parameters` | (Optional) An optional tibble of tuning parameter values that can be used to filter the predicted values before processing. Applies only to `tune_results` objects. |
| `...` | Additional arguments passed to the models or routines used to calculate the new probabilities. |
| `.by` | The column identifier for the grouping variable. This should be a single unquoted column name that selects a qualitative variable for grouping. Default to `NULL`. When `.by = NULL` no grouping will take place. |

## Details

This function uses `stats::isoreg()` to create obtain the calibration values. It runs `stats::isoreg()` multiple times, and each time with a different seed. The results are saved inside the returned `cal_object`.

## Multiclass Extension

This method is designed to work with two classes. For multiclass, it creates a set of "one versus all" calibrations for each class. After they are applied to the data, the probability estimates are re-normalized to add to one. This final step might compromise the calibration.

## See Also

https://www.tidymodels.org/learn/models/calibration/, `cal_validate_isotonic_boot()`

## Examples

```
# It will automatically identify the probability columns
# if passed a model fitted with tidymodels
cal_estimate_isotonic_boot(segment_logistic, Class)
# Specify the variable names in a vector of unquoted names
cal_estimate_isotonic_boot(segment_logistic, Class, c(.pred_poor, .pred_good))
# dplyr selector functions are also supported
cal_estimate_isotonic_boot(segment_logistic, Class, dplyr::starts_with(".pred"))
```

---

# cal_estimate_linear *Uses a linear regression model to calibrate numeric predictions*

---

## Description

Uses a linear regression model to calibrate numeric predictions

## Usage

```
cal_estimate_linear(
  .data,
  truth = NULL,
  estimate = dplyr::matches("^.pred$"),
  smooth = TRUE,
  parameters = NULL,
  ...,
  .by = NULL
)

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

```
  estimate = dplyr::matches("^.pred$"),
  smooth = TRUE,
  parameters = NULL,
  ...,
  .by = NULL
)

## S3 method for class 'tune_results'
cal_estimate_linear(
  .data,
  truth = NULL,
  estimate = dplyr::matches("^.pred$"),
  smooth = TRUE,
  parameters = NULL,
  ...
)

## S3 method for class 'grouped_df'
cal_estimate_linear(
  .data,
  truth = NULL,
  estimate = NULL,
  smooth = TRUE,
  parameters = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `.data` | Am ungrouped `data.frame` object, or `tune_results` object, that contains a prediction column. |
| `truth` | The column identifier for the observed outcome data (that is numeric). This should be an unquoted column name. |
| `estimate` | Column identifier for the predicted values |
| `smooth` | Applies to the linear models. It switches between a generalized additive model using spline terms when `TRUE`, and simple linear regression when `FALSE`. |
| `parameters` | (Optional) An optional tibble of tuning parameter values that can be used to filter the predicted values before processing. Applies only to `tune_results` objects. |
| `...` | Additional arguments passed to the models or routines used to calculate the new predictions. |
| `.by` | The column identifier for the grouping variable. This should be a single unquoted column name that selects a qualitative variable for grouping. Default to `NULL`. When `.by = NULL` no grouping will take place. |

## Details

This function uses existing modeling functions from other packages to create the calibration:

- stats::glm() is used when smooth is set to FALSE
- mgcv::gam() is used when smooth is set to TRUE

These methods estimate the relationship in the unmodified predicted values and then remove that trend when cal_apply() is invoked.

## See Also

https://www.tidymodels.org/learn/models/calibration/, cal_validate_linear()

## Examples

```
library(dplyr)
library(ggplot2)

head(boosting_predictions_test)

# ------------------------------------------------------------------------------
# Before calibration

y_rng <- extendrange(boosting_predictions_test$outcome)

boosting_predictions_test %>%
  ggplot(aes(outcome, .pred)) +
  geom_abline(lty = 2) +
  geom_point(alpha = 1 / 2) +
  geom_smooth(se = FALSE, col = "blue", linewidth = 1.2, alpha = 3 / 4) +
  coord_equal(xlim = y_rng, ylim = y_rng) +
  ggtitle("Before calibration")

# ------------------------------------------------------------------------------
# Smoothed trend removal

smoothed_cal <-
  boosting_predictions_oob %>%
  # It will automatically identify the predicted value columns when the
  # standard tidymodels naming conventions are used.
  cal_estimate_linear(outcome)
smoothed_cal

boosting_predictions_test %>%
  cal_apply(smoothed_cal) %>%
  ggplot(aes(outcome, .pred)) +
  geom_abline(lty = 2) +
  geom_point(alpha = 1 / 2) +
  geom_smooth(se = FALSE, col = "blue", linewidth = 1.2, alpha = 3 / 4) +
  coord_equal(xlim = y_rng, ylim = y_rng) +
  ggtitle("After calibration")
```

`cal_estimate_logistic` *Uses a logistic regression model to calibrate probabilities*

## Description

Uses a logistic regression model to calibrate probabilities

## Usage

```
cal_estimate_logistic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  smooth = TRUE,
  parameters = NULL,
  ...
)

## S3 method for class 'data.frame'
cal_estimate_logistic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  smooth = TRUE,
  parameters = NULL,
  ...,
  .by = NULL
)

## S3 method for class 'tune_results'
cal_estimate_logistic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  smooth = TRUE,
  parameters = NULL,
  ...
)

## S3 method for class 'grouped_df'
cal_estimate_logistic(
  .data,
  truth = NULL,
  estimate = NULL,
  smooth = TRUE,
  parameters = NULL,
  ...
```

)

## Arguments

| | |
|---|---|
| .data | An ungrouped data.frame object, or tune_results object, that contains predictions and probability columns. |
| truth | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| estimate | A vector of column identifiers, or one of dplyr selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (.pred_). The order of the identifiers will be considered the same as the order of the levels of the truth variable. |
| smooth | Applies to the logistic models. It switches between logistic spline when TRUE, and simple logistic regression when FALSE. |
| parameters | (Optional) An optional tibble of tuning parameter values that can be used to filter the predicted values before processing. Applies only to tune_results objects. |
| ... | Additional arguments passed to the models or routines used to calculate the new probabilities. |
| .by | The column identifier for the grouping variable. This should be a single unquoted column name that selects a qualitative variable for grouping. Default to NULL. When .by = NULL no grouping will take place. |

## Details

This function uses existing modeling functions from other packages to create the calibration:

- stats::glm() is used when smooth is set to FALSE
- mgcv::gam() is used when smooth is set to TRUE

**Multiclass Extension:**

This method has *not* been extended to multiclass outcomes. However, the natural multiclass extension is cal_estimate_multinomial().

## See Also

https://www.tidymodels.org/learn/models/calibration/, cal_validate_logistic()

## Examples

```
# It will automatically identify the probability columns
# if passed a model fitted with tidymodels
cal_estimate_logistic(segment_logistic, Class)

# Specify the variable names in a vector of unquoted names
cal_estimate_logistic(segment_logistic, Class, c(.pred_poor, .pred_good))

# dplyr selector functions are also supported
cal_estimate_logistic(segment_logistic, Class, dplyr::starts_with(".pred_"))
```

`cal_estimate_multinomial` *Uses a Multinomial calibration model to calculate new probabilities*

## Description

Uses a Multinomial calibration model to calculate new probabilities

## Usage

```
cal_estimate_multinomial(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  smooth = TRUE,
  parameters = NULL,
  ...
)

## S3 method for class 'data.frame'
cal_estimate_multinomial(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  smooth = TRUE,
  parameters = NULL,
  ...,
  .by = NULL
)

## S3 method for class 'tune_results'
cal_estimate_multinomial(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  smooth = TRUE,
  parameters = NULL,
  ...
)

## S3 method for class 'grouped_df'
cal_estimate_multinomial(
  .data,
  truth = NULL,
  estimate = NULL,
  smooth = TRUE,
  parameters = NULL,
```

```
  ...
)
```

## Arguments

| | |
|---|---|
| .data | An ungrouped data.frame object, or tune_results object, that contains predictions and probability columns. |
| truth | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| estimate | A vector of column identifiers, or one of dplyr selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (.pred_). The order of the identifiers will be considered the same as the order of the levels of the truth variable. |
| smooth | Applies to the logistic models. It switches between logistic spline when TRUE, and simple logistic regression when FALSE. |
| parameters | (Optional) An optional tibble of tuning parameter values that can be used to filter the predicted values before processing. Applies only to tune_results objects. |
| ... | Additional arguments passed to the models or routines used to calculate the new probabilities. |
| .by | The column identifier for the grouping variable. This should be a single unquoted column name that selects a qualitative variable for grouping. Default to NULL. When .by = NULL no grouping will take place. |

## Details

When smooth = FALSE, nnet::multinom() function is used to estimate the model, otherwise mgcv::gam() is used.

## See Also

https://www.tidymodels.org/learn/models/calibration/, cal_validate_multinomial()

## Examples

```
library(modeldata)
library(parsnip)
library(dplyr)

f <-
  list(
    ~ -0.5 + 0.6 * abs(A),
    ~ ifelse(A > 0 & B > 0, 1.0 + 0.2 * A / B, -2),
    ~ -0.6 * A + 0.50 * B - A * B
  )

set.seed(1)
tr_dat <- sim_multinomial(500, eqn_1 = f[[1]], eqn_2 = f[[2]], eqn_3 = f[[3]])
cal_dat <- sim_multinomial(500, eqn_1 = f[[1]], eqn_2 = f[[2]], eqn_3 = f[[3]])
te_dat <- sim_multinomial(500, eqn_1 = f[[1]], eqn_2 = f[[2]], eqn_3 = f[[3]])
```

```
set.seed(2)
rf_fit <-
  rand_forest() %>%
  set_mode("classification") %>%
  set_engine("randomForest") %>%
  fit(class ~ ., data = tr_dat)

cal_pred <-
  predict(rf_fit, cal_dat, type = "prob") %>%
  bind_cols(cal_dat)
te_pred <-
  predict(rf_fit, te_dat, type = "prob") %>%
  bind_cols(te_dat)

cal_plot_windowed(cal_pred, truth = class, window_size = 0.1, step_size = 0.03)

smoothed_mn <- cal_estimate_multinomial(cal_pred, truth = class)

new_test_pred <- cal_apply(te_pred, smoothed_mn)

cal_plot_windowed(new_test_pred, truth = class, window_size = 0.1, step_size = 0.03)
```

---

`cal_plot_breaks` *Probability calibration plots via binning*

---

## Description

A plot is created to assess whether the observed rate of the event is about the same as the predicted probability of the event from some model.

A sequence of even, mutually exclusive bins are created from zero to one. For each bin, the data whose predicted probability falls within the range of the bin is used to calculate the observed event rate (along with confidence intervals for the event rate). If the predictions are well calibrated, the fitted curve should align with the diagonal line.

## Usage

```
cal_plot_breaks(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  num_breaks = 10,
  conf_level = 0.9,
  include_ribbon = TRUE,
  include_rug = TRUE,
  include_points = TRUE,
  event_level = c("auto", "first", "second"),
```

```
  ...
)

## S3 method for class 'data.frame'
cal_plot_breaks(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  num_breaks = 10,
  conf_level = 0.9,
  include_ribbon = TRUE,
  include_rug = TRUE,
  include_points = TRUE,
  event_level = c("auto", "first", "second"),
  ...,
  .by = NULL
)

## S3 method for class 'tune_results'
cal_plot_breaks(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  num_breaks = 10,
  conf_level = 0.9,
  include_ribbon = TRUE,
  include_rug = TRUE,
  include_points = TRUE,
  event_level = c("auto", "first", "second"),
  ...
)

## S3 method for class 'grouped_df'
cal_plot_breaks(
  .data,
  truth = NULL,
  estimate = NULL,
  num_breaks = 10,
  conf_level = 0.9,
  include_ribbon = TRUE,
  include_rug = TRUE,
  include_points = TRUE,
  event_level = c("auto", "first", "second"),
  ...
)
```

## Arguments

| | |
|---|---|
| .data | An ungrouped data frame object containing predictions and probability columns. |

| | |
|---|---|
| truth | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| estimate | A vector of column identifiers, or one of dplyr selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (.pred_). The order of the identifiers will be considered the same as the order of the levels of the truth variable. |
| num_breaks | The number of segments to group the probabilities. It defaults to 10. |
| conf_level | Confidence level to use in the visualization. It defaults to 0.9. |
| include_ribbon | Flag that indicates if the ribbon layer is to be included. It defaults to TRUE. |
| include_rug | Flag that indicates if the Rug layer is to be included. It defaults to TRUE. In the plot, the top side shows the frequency the event occurring, and the bottom the frequency of the event not occurring. |
| include_points | Flag that indicates if the point layer is to be included. |
| event_level | single string. Either "first" or "second" to specify which level of truth to consider as the "event". Defaults to "auto", which allows the function decide which one to use based on the type of model (binary, multi-class or linear) |
| ... | Additional arguments passed to the tune_results object. |
| .by | The column identifier for the grouping variable. This should be a single unquoted column name that selects a qualitative variable for grouping. Default to NULL. When .by = NULL no grouping will take place. |

## Value

A ggplot object.

## See Also

https://www.tidymodels.org/learn/models/calibration/, cal_plot_windowed(), cal_plot_logistic()

cal_plot_logistic(), cal_plot_windowed()

## Examples

```
library(ggplot2)
library(dplyr)

cal_plot_breaks(
  segment_logistic,
  Class,
  .pred_good
)

cal_plot_logistic(
  segment_logistic,
  Class,
  .pred_good
)
```

```
cal_plot_windowed(
  segment_logistic,
  Class,
  .pred_good
)

# The functions support dplyr groups

model <- glm(Class ~ .pred_good, segment_logistic, family = "binomial")

preds <- predict(model, segment_logistic, type = "response")

gl <- segment_logistic %>%
  mutate(.pred_good = 1 - preds, source = "glm")

combined <- bind_rows(mutate(segment_logistic, source = "original"), gl)

combined %>%
  cal_plot_logistic(Class, .pred_good, .by = source)

# The grouping can be faceted in ggplot2
combined %>%
  cal_plot_logistic(Class, .pred_good, .by = source) +
  facet_wrap(~source) +
  theme(legend.position = "")
```

---

`cal_plot_logistic` *Probability calibration plots via logistic regression*

---

## Description

A logistic regression model is fit where the original outcome data are used as the outcome and the estimated class probabilities for one class are used as the predictor. If `smooth = TRUE`, a generalized additive model is fit using `mgcv::gam()` and the default smoothing method. Otherwise, a simple logistic regression is used.

If the predictions are well calibrated, the fitted curve should align with the diagonal line. Confidence intervals for the fitted line are also shown.

## Usage

```
cal_plot_logistic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  conf_level = 0.9,
  smooth = TRUE,
  include_rug = TRUE,
  include_ribbon = TRUE,
  event_level = c("auto", "first", "second"),
```

```
  ...
)

## S3 method for class 'data.frame'
cal_plot_logistic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  conf_level = 0.9,
  smooth = TRUE,
  include_rug = TRUE,
  include_ribbon = TRUE,
  event_level = c("auto", "first", "second"),
  ...,
  .by = NULL
)

## S3 method for class 'tune_results'
cal_plot_logistic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  conf_level = 0.9,
  smooth = TRUE,
  include_rug = TRUE,
  include_ribbon = TRUE,
  event_level = c("auto", "first", "second"),
  ...
)

## S3 method for class 'grouped_df'
cal_plot_logistic(
  .data,
  truth = NULL,
  estimate = NULL,
  conf_level = 0.9,
  smooth = TRUE,
  include_rug = TRUE,
  include_ribbon = TRUE,
  event_level = c("auto", "first", "second"),
  ...
)
```

## Arguments

| | |
|---|---|
| .data | An ungrouped data frame object containing predictions and probability columns. |
| truth | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |

| | |
|---|---|
| estimate | A vector of column identifiers, or one of dplyr selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (.pred_). The order of the identifiers will be considered the same as the order of the levels of the truth variable. |
| conf_level | Confidence level to use in the visualization. It defaults to 0.9. |
| smooth | A logical for using a generalized additive model with smooth terms for the predictor via mgcv::gam() and mgcv::s(). |
| include_rug | Flag that indicates if the Rug layer is to be included. It defaults to TRUE. In the plot, the top side shows the frequency the event occurring, and the bottom the frequency of the event not occurring. |
| include_ribbon | Flag that indicates if the ribbon layer is to be included. It defaults to TRUE. |
| event_level | single string. Either "first" or "second" to specify which level of truth to consider as the "event". Defaults to "auto", which allows the function decide which one to use based on the type of model (binary, multi-class or linear) |
| ... | Additional arguments passed to the tune_results object. |
| .by | The column identifier for the grouping variable. This should be a single unquoted column name that selects a qualitative variable for grouping. Default to NULL. When .by = NULL no grouping will take place. |

## Value

A ggplot object.

## See Also

https://www.tidymodels.org/learn/models/calibration/, cal_plot_windowed(), cal_plot_breaks()

cal_plot_breaks(), cal_plot_windowed()

## Examples

```
library(ggplot2)
library(dplyr)

cal_plot_logistic(
  segment_logistic,
  Class,
  .pred_good
)

cal_plot_logistic(
  segment_logistic,
  Class,
  .pred_good,
  smooth = FALSE
)
```

`cal_plot_regression` *Regression calibration plots*

## Description

A scatter plot of the observed and predicted values is computed where the axes are the same. When `smooth = TRUE`, a generalized additive model fit is shown. If the predictions are well calibrated, the fitted curve should align with the diagonal line.

## Usage

```
cal_plot_regression(.data, truth = NULL, estimate = NULL, smooth = TRUE, ...)

## S3 method for class 'data.frame'
cal_plot_regression(
  .data,
  truth = NULL,
  estimate = NULL,
  smooth = TRUE,
  ...,
  .by = NULL
)

## S3 method for class 'tune_results'
cal_plot_regression(.data, truth = NULL, estimate = NULL, smooth = TRUE, ...)

## S3 method for class 'grouped_df'
cal_plot_regression(.data, truth = NULL, estimate = NULL, smooth = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `.data` | An ungrouped data frame object containing a prediction column. |
| `truth` | The column identifier for the true results (numeric). This should be an unquoted column name. |
| `estimate` | The column identifier for the predictions. This should be an unquoted column name |
| `smooth` | A logical: should a smoother curve be added. |
| `...` | Additional arguments passed to `ggplot2::geom_point()`. |
| `.by` | The column identifier for the grouping variable. This should be a single unquoted column name that selects a qualitative variable for grouping. Default to `NULL`. When `.by = NULL` no grouping will take place. |

## Value

A ggplot object.

## Examples

```
cal_plot_regression(boosting_predictions_oob, outcome, .pred)

cal_plot_regression(boosting_predictions_oob, outcome, .pred,
  alpha = 1 / 6, cex = 3, smooth = FALSE
)

cal_plot_regression(boosting_predictions_oob, outcome, .pred,
  .by = id,
  alpha = 1 / 6, cex = 3, smooth = FALSE
)
```

---

# cal_plot_windowed *Probability calibration plots via moving windows*

---

## Description

A plot is created to assess whether the observed rate of the event is about the sample as the predicted probability of the event from some model. This is similar to `cal_plot_breaks()`, except that the bins are overlapping.

A sequence of bins are created from zero to one. For each bin, the data whose predicted probability falls within the range of the bin is used to calculate the observed event rate (along with confidence intervals for the event rate).

If the predictions are well calibrated, the fitted curve should align with the diagonal line.

## Usage

```
cal_plot_windowed(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  window_size = 0.1,
  step_size = window_size/2,
  conf_level = 0.9,
  include_ribbon = TRUE,
  include_rug = TRUE,
  include_points = TRUE,
  event_level = c("auto", "first", "second"),
  ...
)

## S3 method for class 'data.frame'
cal_plot_windowed(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  window_size = 0.1,
```

```
  step_size = window_size/2,
  conf_level = 0.9,
  include_ribbon = TRUE,
  include_rug = TRUE,
  include_points = TRUE,
  event_level = c("auto", "first", "second"),
  ...,
  .by = NULL
)

## S3 method for class 'tune_results'
cal_plot_windowed(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  window_size = 0.1,
  step_size = window_size/2,
  conf_level = 0.9,
  include_ribbon = TRUE,
  include_rug = TRUE,
  include_points = TRUE,
  event_level = c("auto", "first", "second"),
  ...
)

## S3 method for class 'grouped_df'
cal_plot_windowed(
  .data,
  truth = NULL,
  estimate = NULL,
  window_size = 0.1,
  step_size = window_size/2,
  conf_level = 0.9,
  include_ribbon = TRUE,
  include_rug = TRUE,
  include_points = TRUE,
  event_level = c("auto", "first", "second"),
  ...
)
```

## Arguments

| | |
|---|---|
| .data | An ungrouped data frame object containing predictions and probability columns. |
| truth | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| estimate | A vector of column identifiers, or one of dplyr selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (.pred_). The order of the identifiers will be considered the same as |

| | |
|---|---|
| | the order of the levels of the `truth` variable. |
| `window_size` | The size of segments. Used for the windowed probability calculations. It defaults to 10% of segments. |
| `step_size` | The gap between segments. Used for the windowed probability calculations. It defaults to half the size of `window_size` |
| `conf_level` | Confidence level to use in the visualization. It defaults to 0.9. |
| `include_ribbon` | Flag that indicates if the ribbon layer is to be included. It defaults to `TRUE`. |
| `include_rug` | Flag that indicates if the Rug layer is to be included. It defaults to `TRUE`. In the plot, the top side shows the frequency the event occurring, and the bottom the frequency of the event not occurring. |
| `include_points` | Flag that indicates if the point layer is to be included. |
| `event_level` | single string. Either "first" or "second" to specify which level of truth to consider as the "event". Defaults to "auto", which allows the function decide which one to use based on the type of model (binary, multi-class or linear) |
| `...` | Additional arguments passed to the `tune_results` object. |
| `.by` | The column identifier for the grouping variable. This should be a single unquoted column name that selects a qualitative variable for grouping. Default to `NULL`. When `.by = NULL` no grouping will take place. |

## Value

A ggplot object.

## See Also

https://www.tidymodels.org/learn/models/calibration/, `cal_plot_logistic()`, `cal_plot_breaks()`

`cal_plot_breaks()`, `cal_plot_logistic()`

## Examples

```
library(ggplot2)
library(dplyr)

cal_plot_windowed(
  segment_logistic,
  Class,
  .pred_good
)

# More breaks
cal_plot_windowed(
  segment_logistic,
  Class,
  .pred_good,
  window_size = 0.05
)
```

`cal_validate_beta` *Measure performance with and without using Beta calibration*

## Description

This function uses resampling to measure the effect of calibrating predicted values.

## Usage

```
cal_validate_beta(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'resample_results'
cal_validate_beta(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'rset'
cal_validate_beta(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'tune_results'
cal_validate_beta(
  .data,
  truth = NULL,
  estimate = NULL,
  metrics = NULL,
  save_pred = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| .data | An rset object or the results of tune::fit_resamples() with a .predictions column. |
| truth | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| estimate | A vector of column identifiers, or one of dplyr selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (.pred_). The order of the identifiers will be considered the same as the order of the levels of the truth variable. |
| metrics | A set of metrics passed created via yardstick::metric_set() |
| save_pred | Indicates whether to a column of post-calibration predictions. |
| ... | Options to pass to cal_estimate_beta(), such as the shape_params and location_params arguments. |

## Details

These functions are designed to calculate performance with and without calibration. They use resampling to measure out-of-sample effectiveness. There are two ways to pass the data in:

- If you have a data frame of predictions, an rset object can be created via **rsample** functions. See the example below.
- If you have already made a resampling object from the original data and used it with tune::fit_resamples(), you can pass that object to the calibration function and it will use the same resampling scheme. If a different resampling scheme should be used, run tune::collect_predictions() on the object and use the process in the previous bullet point.

Please note that these functions do not apply to tune_result objects. The notion of "validation" implies that the tuning parameter selection has been resolved.

collect_predictions() can be used to aggregate the metrics for analysis.

## Value

The original object with a .metrics_cal column and, optionally, an additional .predictions_cal column. The class cal_rset is also added.

## Performance Metrics

By default, the average of the Brier scores is returned. Any appropriate yardstick::metric_set() can be used. The validation function compares the average of the metrics before, and after the calibration.

## See Also

https://www.tidymodels.org/learn/models/calibration/, cal_estimate_beta()

## Examples

```
library(dplyr)

segment_logistic %>%
  rsample::vfold_cv() %>%
  cal_validate_beta(Class)
```

---

`cal_validate_isotonic` *Measure performance with and without using isotonic regression calibration*

---

## Description

This function uses resampling to measure the effect of calibrating predicted values.

## Usage

```
cal_validate_isotonic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'resample_results'
cal_validate_isotonic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'rset'
cal_validate_isotonic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)
```

```
## S3 method for class 'tune_results'
cal_validate_isotonic(
  .data,
  truth = NULL,
  estimate = NULL,
  metrics = NULL,
  save_pred = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| .data | An rset object or the results of tune::fit_resamples() with a .predictions column. |
| truth | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| estimate | A vector of column identifiers, or one of dplyr selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (.pred_). The order of the identifiers will be considered the same as the order of the levels of the truth variable. |
| metrics | A set of metrics passed created via yardstick::metric_set() |
| save_pred | Indicates whether to a column of post-calibration predictions. |
| ... | Options to pass to cal_estimate_logistic(), such as the smooth argument. |

## Details

These functions are designed to calculate performance with and without calibration. They use resampling to measure out-of-sample effectiveness. There are two ways to pass the data in:

- If you have a data frame of predictions, an rset object can be created via **rsample** functions. See the example below.
- If you have already made a resampling object from the original data and used it with tune::fit_resamples(), you can pass that object to the calibration function and it will use the same resampling scheme. If a different resampling scheme should be used, run tune::collect_predictions() on the object and use the process in the previous bullet point.

Please note that these functions do not apply to tune_result objects. The notion of "validation" implies that the tuning parameter selection has been resolved.

collect_predictions() can be used to aggregate the metrics for analysis.

## Value

The original object with a .metrics_cal column and, optionally, an additional .predictions_cal column. The class cal_rset is also added.

### Performance Metrics

By default, the average of the Brier scores (classification calibration) or the root mean squared error (regression) is returned. Any appropriate `yardstick::metric_set()` can be used. The validation function compares the average of the metrics before, and after the calibration.

### See Also

https://www.tidymodels.org/learn/models/calibration/, `cal_estimate_isotonic()`

### Examples

```
library(dplyr)

segment_logistic %>%
  rsample::vfold_cv() %>%
  cal_validate_isotonic(Class)
```

---

## cal_validate_isotonic_boot

*Measure performance with and without using bagged isotonic regression calibration*

---

### Description

This function uses resampling to measure the effect of calibrating predicted values.

### Usage

```
cal_validate_isotonic_boot(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'resample_results'
cal_validate_isotonic_boot(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)
```

```
## S3 method for class 'rset'
cal_validate_isotonic_boot(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'tune_results'
cal_validate_isotonic_boot(
  .data,
  truth = NULL,
  estimate = NULL,
  metrics = NULL,
  save_pred = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `.data` | An `rset` object or the results of `tune::fit_resamples()` with a `.predictions` column. |
| `truth` | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| `estimate` | A vector of column identifiers, or one of `dplyr` selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (`.pred_`). The order of the identifiers will be considered the same as the order of the levels of the `truth` variable. |
| `metrics` | A set of metrics passed created via `yardstick::metric_set()` |
| `save_pred` | Indicates whether to a column of post-calibration predictions. |
| `...` | Options to pass to `cal_estimate_isotonic_boot()`, such as the `times` argument. |

## Details

These functions are designed to calculate performance with and without calibration. They use resampling to measure out-of-sample effectiveness. There are two ways to pass the data in:

- If you have a data frame of predictions, an `rset` object can be created via **rsample** functions. See the example below.
- If you have already made a resampling object from the original data and used it with `tune::fit_resamples()`, you can pass that object to the calibration function and it will use the same resampling scheme. If a different resampling scheme should be used, run `tune::collect_predictions()` on the object and use the process in the previous bullet point.

Please note that these functions do not apply to `tune_result` objects. The notion of "validation" implies that the tuning parameter selection has been resolved.

`collect_predictions()` can be used to aggregate the metrics for analysis.

## Value

The original object with a `.metrics_cal` column and, optionally, an additional `.predictions_cal` column. The class `cal_rset` is also added.

## Performance Metrics

By default, the average of the Brier scores (classification calibration) or the root mean squared error (regression) is returned. Any appropriate `yardstick::metric_set()` can be used. The validation function compares the average of the metrics before, and after the calibration.

## See Also

https://www.tidymodels.org/learn/models/calibration/, `cal_estimate_isotonic_boot()`

## Examples

```
library(dplyr)

segment_logistic %>%
  rsample::vfold_cv() %>%
  cal_validate_isotonic_boot(Class)
```

---

# cal_validate_linear *Measure performance with and without using linear regression calibration*

---

## Description

Measure performance with and without using linear regression calibration

## Usage

```
cal_validate_linear(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'resample_results'
```

```
cal_validate_linear(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'rset'
cal_validate_linear(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `.data` | An `rset` object or the results of `tune::fit_resamples()` with a `.predictions` column. |
| `truth` | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| `estimate` | A vector of column identifiers, or one of `dplyr` selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (`.pred_`). The order of the identifiers will be considered the same as the order of the levels of the `truth` variable. |
| `metrics` | A set of metrics passed created via `yardstick::metric_set()` |
| `save_pred` | Indicates whether to a column of post-calibration predictions. |
| `...` | Options to pass to `cal_estimate_logistic()`, such as the `smooth` argument. |

## Performance Metrics

By default, the average of the root mean square error (RMSE) is returned. Any appropriate `yardstick::metric_set()` can be used. The validation function compares the average of the metrics before, and after the calibration.

## See Also

https://www.tidymodels.org/learn/models/calibration/, `cal_estimate_linear()`

## Examples

```
library(dplyr)
library(yardstick)
library(rsample)
```

```
head(boosting_predictions_test)

reg_stats <- metric_set(rmse, ccc)

set.seed(828)
boosting_predictions_oob %>%
  # Resample with 10-fold cross-validation
  vfold_cv() %>%
  cal_validate_linear(truth = outcome, smooth = FALSE, metrics = reg_stats)
```

---

`cal_validate_logistic` *Measure performance with and without using logistic calibration*

---

## Description

This function uses resampling to measure the effect of calibrating predicted values.

## Usage

```
cal_validate_logistic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'resample_results'
cal_validate_logistic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'rset'
cal_validate_logistic(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)
```

```
## S3 method for class 'tune_results'
cal_validate_logistic(
  .data,
  truth = NULL,
  estimate = NULL,
  metrics = NULL,
  save_pred = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| .data | An rset object or the results of tune::fit_resamples() with a .predictions column. |
| truth | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| estimate | A vector of column identifiers, or one of dplyr selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (.pred_). The order of the identifiers will be considered the same as the order of the levels of the truth variable. |
| metrics | A set of metrics passed created via yardstick::metric_set() |
| save_pred | Indicates whether to a column of post-calibration predictions. |
| ... | Options to pass to cal_estimate_logistic(), such as the smooth argument. |

## Details

These functions are designed to calculate performance with and without calibration. They use resampling to measure out-of-sample effectiveness. There are two ways to pass the data in:

- If you have a data frame of predictions, an rset object can be created via **rsample** functions. See the example below.
- If you have already made a resampling object from the original data and used it with tune::fit_resamples(), you can pass that object to the calibration function and it will use the same resampling scheme. If a different resampling scheme should be used, run tune::collect_predictions() on the object and use the process in the previous bullet point.

Please note that these functions do not apply to tune_result objects. The notion of "validation" implies that the tuning parameter selection has been resolved.

collect_predictions() can be used to aggregate the metrics for analysis.

## Value

The original object with a .metrics_cal column and, optionally, an additional .predictions_cal column. The class cal_rset is also added.

### Performance Metrics

By default, the average of the Brier scores is returned. Any appropriate `yardstick::metric_set()` can be used. The validation function compares the average of the metrics before, and after the calibration.

### See Also

https://www.tidymodels.org/learn/models/calibration/, `cal_estimate_logistic()`

### Examples

```
library(dplyr)

# ------------------------------------------------------------------------------
# classification example

segment_logistic %>%
  rsample::vfold_cv() %>%
  cal_validate_logistic(Class)
```

---

## cal_validate_multinomial

*Measure performance with and without using multinomial calibration*

---

### Description

This function uses resampling to measure the effect of calibrating predicted values.

### Usage

```
cal_validate_multinomial(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'resample_results'
cal_validate_multinomial(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  metrics = NULL,
  save_pred = FALSE,
```

```
  ...
)

## S3 method for class 'rset'
cal_validate_multinomial(
  .data,
  truth = NULL,
  estimate = dplyr::starts_with(".pred_"),
  metrics = NULL,
  save_pred = FALSE,
  ...
)

## S3 method for class 'tune_results'
cal_validate_multinomial(
  .data,
  truth = NULL,
  estimate = NULL,
  metrics = NULL,
  save_pred = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `.data` | An `rset` object or the results of `tune::fit_resamples()` with a `.predictions` column. |
| `truth` | The column identifier for the true class results (that is a factor). This should be an unquoted column name. |
| `estimate` | A vector of column identifiers, or one of `dplyr` selector functions to choose which variables contains the class probabilities. It defaults to the prefix used by tidymodels (`.pred_`). The order of the identifiers will be considered the same as the order of the levels of the `truth` variable. |
| `metrics` | A set of metrics passed created via `yardstick::metric_set()` |
| `save_pred` | Indicates whether to a column of post-calibration predictions. |
| `...` | Options to pass to `cal_estimate_logistic()`, such as the `smooth` argument. |

## Details

These functions are designed to calculate performance with and without calibration. They use resampling to measure out-of-sample effectiveness. There are two ways to pass the data in:

- If you have a data frame of predictions, an `rset` object can be created via **rsample** functions. See the example below.
- If you have already made a resampling object from the original data and used it with `tune::fit_resamples()`, you can pass that object to the calibration function and it will use the same resampling scheme. If a different resampling scheme should be used, run `tune::collect_predictions()` on the object and use the process in the previous bullet point.

Please note that these functions do not apply to `tune_result` objects. The notion of "validation" implies that the tuning parameter selection has been resolved.

`collect_predictions()` can be used to aggregate the metrics for analysis.

## Value

The original object with a `.metrics_cal` column and, optionally, an additional `.predictions_cal` column. The class `cal_rset` is also added.

## Performance Metrics

By default, the average of the Brier scores is returned. Any appropriate `yardstick::metric_set()` can be used. The validation function compares the average of the metrics before, and after the calibration.

## See Also

`cal_apply()`, `cal_estimate_multinomial()`

## Examples

```
library(dplyr)

species_probs %>%
  rsample::vfold_cv() %>%
  cal_validate_multinomial(Species)
```

---

# class_pred *Create a class prediction object*

---

## Description

`class_pred()` creates a `class_pred` object from a factor or ordered factor. You can optionally specify values of the factor to be set as *equivocal*.

## Usage

```
class_pred(x = factor(), which = integer(), equivocal = "[EQ]")
```

## Arguments

| | |
|---|---|
| `x` | A factor or ordered factor. |
| `which` | An integer vector specifying the locations of `x` to declare as equivocal. |
| `equivocal` | A single character specifying the equivocal label used when printing. |

## Details

Equivocal values are those that you feel unsure about, and would like to exclude from performance calculations or other metrics.

## Examples

```
x <- factor(c("Yes", "No", "Yes", "Yes"))

# Create a class_pred object from a factor
class_pred(x)

# Say you aren't sure about that 2nd "Yes" value. You could mark it as
# equivocal.
class_pred(x, which = 3)

# Maybe you want a different equivocal label
class_pred(x, which = 3, equivocal = "eq_value")
```

---

# collect_metrics.cal_rset

*Obtain and format metrics produced by calibration validation*

---

## Description

Obtain and format metrics produced by calibration validation

## Usage

```
## S3 method for class 'cal_rset'
collect_metrics(x, summarize = TRUE, ...)
```

## Arguments

| | |
|---|---|
| x | An object produced by one of the validation function (or class `cal_rset`). |
| summarize | A logical; should metrics be summarized over resamples (`TRUE`) or return the values for each individual resample. See `tune::collect_metrics()` for more details. |
| ... | Not currently used. |

## Value

A tibble

## collect_predictions.cal_rset

*Obtain and format predictions produced by calibration validation*

### Description

Obtain and format predictions produced by calibration validation

### Usage

```
## S3 method for class 'cal_rset'
collect_predictions(x, summarize = TRUE, ...)
```

### Arguments

| | |
|---|---|
| x | An object produced by one of the validation function (or class `cal_rset`). |
| summarize | A logical; should predictions be summarized over resamples (`TRUE`) or return the values for each individual resample. See `tune::collect_predictions()` for more details. |
| ... | Not currently used. |

### Value

A tibble

## control_conformal_full

*Controlling the numeric details for conformal inference*

### Description

Controlling the numeric details for conformal inference

### Usage

```
control_conformal_full(
  method = "iterative",
  trial_points = 100,
  var_multiplier = 10,
  max_iter = 100,
  tolerance = .Machine$double.eps^0.25,
  progress = FALSE,
  required_pkgs = character(0),
  seed = sample.int(10^5, 1)
)
```

### Arguments

| | |
|---|---|
| method | The method for computing the intervals. The options are 'search' (using stats::uniroot(), and 'grid'. |
| trial_points | When method = "grid", how many points should be evaluated? |
| var_multiplier | A multiplier for the variance model that determines the possible range of the bounds. |
| max_iter | When method = "iterative", the maximum number of iterations. |
| tolerance | Tolerance value passed to all.equal() to determine convergence during the search computations. |
| progress | Should a progress bar be used to track execution? |
| required_pkgs | An optional character string for which packages are required. |
| seed | A single integer used to control randomness when models are (re)fit. |

### Value

A list object with the options given by the user.

---

## int_conformal_cv *Prediction intervals via conformal inference CV+*

---

### Description

Nonparametric prediction intervals can be computed for fitted regression workflow objects using the CV+ conformal inference method described by Barber *at al* (2018).

### Usage

```
int_conformal_cv(object, ...)

## Default S3 method:
int_conformal_cv(object, ...)

## S3 method for class 'resample_results'
int_conformal_cv(object, ...)

## S3 method for class 'tune_results'
int_conformal_cv(object, parameters, ...)
```

### Arguments

| | |
|---|---|
| object | An object from a tidymodels resampling or tuning function such as tune::fit_resamples(), tune::tune_grid(), or similar. The object should have been produced in a way that the .extracts column contains the fitted workflow for each resample (see the Details below). |

| | |
|---|---|
| `...` | Not currently used. |
| `parameters` | An tibble of tuning parameter values that can be used to filter the predicted values before processing. This tibble should select a single set of hyper-parameter values from the tuning results. This is only required when a tuning object is passed to `object`. |

## Details

This function implements the CV+ method found in Section 3 of Barber *at al* (2018). It uses the resampled model fits and their associated holdout residuals to make prediction intervals for regression models.

This function prepares the objects for the computations. The `predict()` method computes the intervals for new data.

This method was developed for V-fold cross-validation (no repeats). Interval coverage is unknown for any other resampling methods. The function will not stop the computations for other types of resamples, but we have no way of knowing whether the results are appropriate.

## Value

An object of class `"int_conformal_cv"` containing the information to create intervals. The `predict()` method is used to produce the intervals.

## References

Rina Foygel Barber, Emmanuel J. Candès, Aaditya Ramdas, Ryan J. Tibshirani "Predictive inference with the jackknife+," *The Annals of Statistics*, 49(1), 486-507, 2021

## See Also

`predict.int_conformal_cv()`

## Examples

```
library(workflows)
library(dplyr)
library(parsnip)
library(rsample)
library(tune)
library(modeldata)

set.seed(2)
sim_train <- sim_regression(200)
sim_new <- sim_regression(5) %>% select(-outcome)

sim_rs <- vfold_cv(sim_train)

# We'll use a neural network model
mlp_spec <-
  mlp(hidden_units = 5, penalty = 0.01) %>%
  set_mode("regression")
```

```
# Use a control function that saves the predictions as well as the models.
# Consider using the butcher package in the extracts function to have smaller
# object sizes

ctrl <- control_resamples(save_pred = TRUE, extract = I)

set.seed(3)
nnet_res <-
  mlp_spec %>%
  fit_resamples(outcome ~ ., resamples = sim_rs, control = ctrl)

nnet_int_obj <- int_conformal_cv(nnet_res)
nnet_int_obj

predict(nnet_int_obj, sim_new)
```

---

`int_conformal_full` *Prediction intervals via conformal inference*

---

## Description

Nonparametric prediction intervals can be computed for fitted workflow objects using the conformal inference method described by Lei *at al* (2018).

## Usage

```
int_conformal_full(object, ...)

## Default S3 method:
int_conformal_full(object, ...)

## S3 method for class 'workflow'
int_conformal_full(object, train_data, ..., control = control_conformal_full())
```

## Arguments

| | |
|---|---|
| `object` | A fitted `workflows::workflow()` object. |
| `...` | Not currently used. |
| `train_data` | A data frame with the *original predictor data* used to create the fitted workflow (predictors and outcomes). If the workflow does not contain these values, pass them here. If the workflow used a recipe, this should be the data that were inputs to the recipe (and not the product of a recipe). |
| `control` | A control object from `control_conformal_full()` with the numeric minutiae. |

## Details

This function implements what is usually called "full conformal inference" (see Algorithm 1 in Lei *et al* (2018)) since it uses the entire training set to compute the intervals.

This function prepares the objects for the computations. The `predict()` method computes the intervals for new data.

For a given new_data observation, the predictors are appended to the original training set. Then, different "trial" values of the outcome are substituted in for that observation's outcome and the model is re-fit. From each model, the residual associated with the trial value is compared to a quantile of the distribution of the other residuals. Usually the absolute values of the residuals are used. Once the residual of a trial value exceeds the distributional quantile, the value is one of the bounds.

The literature proposed using a grid search of trial values to find the two points that correspond to the prediction intervals. To use this approach, set `method = "grid"` in `control_conformal_full()`. However, the default method `"search` uses two different one-dimensional iterative searches on either side of the predicted value to find values that correspond to the prediction intervals.

For medium to large data sets, the iterative search method is likely to generate slightly smaller intervals. For small training sets, grid search is more likely to have somewhat smaller intervals (and will be more stable). Otherwise, the iterative search method is more precise and several folds faster.

To determine a range of possible values of the intervals, used by both methods, the initial set of training set residuals are modeled using a Gamma generalized linear model with a log link (see the reference by Aitkin below). For a new sample, the absolute size of the residual is estimated and a multiple of this value is computed as an initial guess of the search boundaries.

**Speed:**

The time it takes to compute the intervals depends on the training set size, search parameters (i.e., convergence criterion, number of iterations), the grid size, and the number of worker processes that are used. For the last item, the computations can be parallelized using the future and furrr packages.

To use parallelism, the `future::plan()` function can be invoked to create a parallel backend. For example, let's make an initial workflow:

```
library(tidymodels)
library(probably)
library(future)

tidymodels_prefer()

## Make a fitted workflow from some simulated data:
set.seed(121)
train_dat <- sim_regression(200)
new_dat   <- sim_regression(  5) %>% select(-outcome)

lm_fit <-
  workflow() %>%
  add_model(linear_reg()) %>%
  add_formula(outcome ~ .) %>%
  fit(data = train_dat)
```

```
# Create the object to be used to make prediction intervals
lm_conform <- int_conformal_full(lm_fit, train_dat)
```

We'll use a "multisession" parallel processing plan to compute the intervals for the five new samples in parallel:

```
plan("multisession")

# This is run in parallel:
predict(lm_conform, new_dat)
```

```
## # A tibble: 5 x 2
##   .pred_lower .pred_upper
##         <dbl>       <dbl>
## 1       -17.9        59.6
## 2       -33.7        51.1
## 3       -30.6        48.2
## 4       -17.3        59.6
## 5       -23.3        55.2
```

Using simulations, there are slightly sub-linear speed-ups when using parallel processing to compute the row-wise intervals.

In comparison with parametric intervals:

```
predict(lm_fit, new_dat, type = "pred_int")
```

```
## # A tibble: 5 x 2
##   .pred_lower .pred_upper
##         <dbl>       <dbl>
## 1       -19.2        59.1
## 2       -31.8        49.7
## 3       -31.0        47.6
## 4       -17.8        60.1
## 5       -23.6        54.3
```

## Value

An object of class "int_conformal_full" containing the information to create intervals (which includes the training set data). The `predict()` method is used to produce the intervals.

## References

Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani and Larry Wasserman (2018) Distribution-Free Predictive Inference for Regression, *Journal of the American Statistical Association*, 113:523, 1094-1111

Murray Aitkin, Modelling Variance Heterogeneity in Normal Regression Using GLIM, *Journal of the Royal Statistical Society Series C: Applied Statistics*, Volume 36, Issue 3, November 1987, Pages 332–339.

## See Also

predict.int_conformal_full()

---

int_conformal_quantile *Prediction intervals via conformal inference and quantile regression*

---

## Description

Nonparametric prediction intervals can be computed for fitted regression workflow objects using the split conformal inference method described by Romano *et al* (2019). To compute quantiles, this function uses Quantile Random Forests instead of classic quantile regression.

## Usage

```
int_conformal_quantile(object, ...)

## S3 method for class 'workflow'
int_conformal_quantile(object, train_data, cal_data, level = 0.95, ...)
```

## Arguments

| | |
|---|---|
| object | A fitted workflows::workflow() object. |
| ... | Options to pass to quantregForest::quantregForest() (such as the number of trees). |
| train_data, cal_data | Data frames with the *predictor and outcome data*. train_data should be the same data used to produce object and cal_data is used to produce predictions (and residuals). If the workflow used a recipe, these should be the data that were inputs to the recipe (and not the product of a recipe). |
| level | The confidence level for the intervals. |

## Details

Note that the significance level should be specified in this function (instead of the predict() method).

cal_data should be large enough to get a good estimates of a extreme quantile (e.g., the 95th for 95% interval) and should not include rows that were in the original training set.

Note that the because of the method used to construct the interval, it is possible that the prediction intervals will not include the predicted value.

## Value

An object of class "int_conformal_quantile" containing the information to create intervals (which includes object). The predict() method is used to produce the intervals.

## References

Romano, Yaniv, Evan Patterson, and Emmanuel Candes. "Conformalized quantile regression." *Advances in neural information processing systems* 32 (2019).

## See Also

predict.int_conformal_quantile()

## Examples

```
library(workflows)
library(dplyr)
library(parsnip)
library(rsample)
library(tune)
library(modeldata)

set.seed(2)
sim_train <- sim_regression(500)
sim_cal <- sim_regression(200)
sim_new <- sim_regression(5) %>% select(-outcome)

# We'll use a neural network model
mlp_spec <-
  mlp(hidden_units = 5, penalty = 0.01) %>%
  set_mode("regression")

mlp_wflow <-
  workflow() %>%
  add_model(mlp_spec) %>%
  add_formula(outcome ~ .)

mlp_fit <- fit(mlp_wflow, data = sim_train)

mlp_int <- int_conformal_quantile(mlp_fit, sim_train, sim_cal,
  level = 0.90
)
mlp_int

predict(mlp_int, sim_new)
```

---

# int_conformal_split *Prediction intervals via split conformal inference*

---

## Description

Nonparametric prediction intervals can be computed for fitted regression workflow objects using the split conformal inference method described by Lei *et al* (2018).

## Usage

```
int_conformal_split(object, ...)

## Default S3 method:
int_conformal_split(object, ...)

## S3 method for class 'workflow'
int_conformal_split(object, cal_data, ...)
```

## Arguments

| | |
|---|---|
| object | A fitted workflows::workflow() object. |
| ... | Not currently used. |
| cal_data | A data frame with the *original predictor and outcome data* used to produce predictions (and residuals). If the workflow used a recipe, this should be the data that were inputs to the recipe (and not the product of a recipe). |

## Details

This function implements what is usually called "split conformal inference" (see Algorithm 1 in Lei *et al* (2018)).

This function prepares the statistics for the interval computations. The predict() method computes the intervals for new data and the signficance level is specified there.

cal_data should be large enough to get a good estimates of a extreme quantile (e.g., the 95th for 95% interval) and should not include rows that were in the original training set.

## Value

An object of class "int_conformal_split" containing the information to create intervals (which includes object). The predict() method is used to produce the intervals.

## References

Lei, Jing, et al. "Distribution-free predictive inference for regression." *Journal of the American Statistical Association* 113.523 (2018): 1094-1111.

## See Also

predict.int_conformal_split()

## Examples

```
library(workflows)
library(dplyr)
library(parsnip)
library(rsample)
library(tune)
library(modeldata)
```

```
set.seed(2)
sim_train <- sim_regression(500)
sim_cal <- sim_regression(200)
sim_new <- sim_regression(5) %>% select(-outcome)

# We'll use a neural network model
mlp_spec <-
  mlp(hidden_units = 5, penalty = 0.01) %>%
  set_mode("regression")

mlp_wflow <-
  workflow() %>%
  add_model(mlp_spec) %>%
  add_formula(outcome ~ .)

mlp_fit <- fit(mlp_wflow, data = sim_train)

mlp_int <- int_conformal_split(mlp_fit, sim_cal)
mlp_int

predict(mlp_int, sim_new, level = 0.90)
```

---

`is_class_pred` *Test if an object inherits from* `class_pred`

---

## Description

`is_class_pred()` checks if an object is a `class_pred` object.

## Usage

```
is_class_pred(x)
```

## Arguments

| | |
|---|---|
| `x` | An object. |

## Examples

```
x <- class_pred(factor(1:5))

is_class_pred(x)
```

## levels.class_pred *Extract* class_pred *levels*

### Description

The levels of a class_pred object do *not* include the equivocal value.

### Usage

```
## S3 method for class 'class_pred'
levels(x)
```

### Arguments

| | |
|---|---|
| x | A class_pred object. |

### Examples

```
x <- class_pred(factor(1:5), which = 1)

# notice that even though `1` is not in the `class_pred` vector, the
# level remains from the original factor
levels(x)
```

## locate-equivocal *Locate equivocal values*

### Description

These functions provide multiple methods of checking for equivocal values, and finding their locations.

### Usage

```
is_equivocal(x)

which_equivocal(x)

any_equivocal(x)
```

### Arguments

| | |
|---|---|
| x | A class_pred object. |

## Value

`is_equivocal()` returns a logical vector the same length as `x` where `TRUE` means the value is equivocal.

`which_equivocal()` returns an integer vector specifying the locations of the equivocal values.

`any_equivocal()` returns `TRUE` if there are any equivocal values.

## Examples

```
x <- class_pred(factor(1:10), which = c(2, 5))

is_equivocal(x)

which_equivocal(x)

any_equivocal(x)
```

---

# make_class_pred *Create a* `class_pred` *vector from class probabilities*

---

## Description

These functions can be used to convert class probability estimates to `class_pred` objects with an optional equivocal zone.

## Usage

```
make_class_pred(..., levels, ordered = FALSE, min_prob = 1/length(levels))

make_two_class_pred(
  estimate,
  levels,
  threshold = 0.5,
  ordered = FALSE,
  buffer = NULL
)
```

## Arguments

| | |
|---|---|
| `...` | Numeric vectors corresponding to class probabilities. There should be one for each level in `levels`, and *it is assumed that the vectors are in the same order as* `levels`. |
| `levels` | A character vector of class levels. The length should be the same as the number of selections made through `...`, or length `2` for `make_two_class_pred()`. |
| `ordered` | A single logical to determine if the levels should be regarded as ordered (in the order given). This results in a `class_pred` object that is flagged as ordered. |

| | |
|---|---|
| `min_prob` | A single numeric value. If any probabilities are less than this value (by row), the row is marked as *equivocal*. |
| `estimate` | A single numeric vector corresponding to the class probabilities of the first level in `levels`. |
| `threshold` | A single numeric value for the threshold to call a row to be labeled as the first value of `levels`. |
| `buffer` | A numeric vector of length 1 or 2 for the buffer around `threshold` that defines the equivocal zone (i.e., `threshold - buffer[1]` to `threshold + buffer[2]`). A length 1 vector is recycled to length 2. The default, `NULL`, is interpreted as no equivocal zone. |

## Value

A vector of class `class_pred`.

## Examples

```
library(dplyr)

good <- segment_logistic$.pred_good
lvls <- levels(segment_logistic$Class)

# Equivocal zone of .5 +/- .15
make_two_class_pred(good, lvls, buffer = 0.15)

# Equivocal zone of c(.5 - .05, .5 + .15)
make_two_class_pred(good, lvls, buffer = c(0.05, 0.15))

# These functions are useful alongside dplyr::mutate()
segment_logistic %>%
  mutate(
    .class_pred = make_two_class_pred(
      estimate = .pred_good,
      levels = levels(Class),
      buffer = 0.15
    )
  )

# Multi-class example
# Note that we provide class probability columns in the same
# order as the levels
species_probs %>%
  mutate(
    .class_pred = make_class_pred(
      .pred_bobcat, .pred_coyote, .pred_gray_fox,
      levels = levels(Species),
      min_prob = .5
    )
  )
```

predict.int_conformal_full *Prediction intervals from conformal methods*

## Description

Prediction intervals from conformal methods

## Usage

```
## S3 method for class 'int_conformal_full'
predict(object, new_data, level = 0.95, ...)

## S3 method for class 'int_conformal_cv'
predict(object, new_data, level = 0.95, ...)

## S3 method for class 'int_conformal_quantile'
predict(object, new_data, ...)

## S3 method for class 'int_conformal_split'
predict(object, new_data, level = 0.95, ...)
```

## Arguments

| | |
|---|---|
| object | An object produced by predict.int_conformal_full(). |
| new_data | A data frame of predictors. |
| level | The confidence level for the intervals. |
| ... | Not currently used. |

## Details

For the CV+. estimator produced by int_conformal_cv(), the intervals are centered around the mean of the predictions produced by the resample-specific model. For example, with 10-fold cross-validation, .pred is the average of the predictions from the 10 models produced by each fold. This may differ from the prediction generated from a model fit that was trained on the entire training set, especially if the training sets are small.

## Value

A tibble with columns .pred_lower and .pred_upper. If the computations for the prediction bound fail, a missing value is used. For objects produced by int_conformal_cv(), an additional .pred column is also returned (see Details below).

## See Also

int_conformal_full(), int_conformal_cv()

## reportable_rate *Calculate the reportable rate*

### Description

The *reportable rate* is defined as the percentage of class predictions that are *not* equivocal.

### Usage

```
reportable_rate(x)
```

### Arguments

| | |
|---|---|
| x | A class_pred object. |

### Details

The reportable rate is calculated as (n_not_equivocal / n).

### Examples

```
x <- class_pred(factor(1:5), which = c(1, 2))

# 3 / 5
reportable_rate(x)
```

## segment_naive_bayes *Image segmentation predictions*

### Description

Image segmentation predictions

### Details

These objects contain test set predictions for the cell segmentation data from Hill, LaPan, Li and Haney (2007). Each data frame are the results from different models (naive Bayes and logistic regression).

### Value

| | |
|---|---|
| segment_naive_bayes, segment_logistic | a tibble |

## Source

Hill, LaPan, Li and Haney (2007). Impact of image segmentation on high-content screening data quality for SK-BR-3 cells, *BMC Bioinformatics*, Vol. 8, pg. 340, https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-340.

## Examples

```
data(segment_naive_bayes)
data(segment_logistic)
```

---

# species_probs *Predictions on animal species*

---

## Description

Predictions on animal species

## Details

These data are holdout predictions from resampling for the animal scat data of Reid (2015) based on a C5.0 classification model.

## Value

`species_probs` a tibble

## Source

Reid, R. E. B. (2015). A morphometric modeling approach to distinguishing among bobcat, coyote and gray fox scats. *Wildlife Biology*, 21(5), 254-262

## Examples

```
data(species_probs)
str(species_probs)
```

threshold_perf *Generate performance metrics across probability thresholds*

## Description

threshold_perf() can take a set of class probability predictions and determine performance characteristics across different values of the probability threshold and any existing groups.

## Usage

```
threshold_perf(.data, ...)

## S3 method for class 'data.frame'
threshold_perf(
  .data,
  truth,
  estimate,
  thresholds = NULL,
  metrics = NULL,
  na_rm = TRUE,
  event_level = "first",
  ...
)
```

## Arguments

| | |
|---|---|
| .data | A tibble, potentially grouped. |
| ... | Currently unused. |
| truth | The column identifier for the true two-class results (that is a factor). This should be an unquoted column name. |
| estimate | The column identifier for the predicted class probabilities (that is a numeric). This should be an unquoted column name. |
| thresholds | A numeric vector of values for the probability threshold. If unspecified, a series of values between 0.5 and 1.0 are used. **Note**: if this argument is used, it must be named. |
| metrics | Either NULL or a yardstick::metric_set() with a list of performance metrics to calculate. The metrics should all be oriented towards hard class predictions (e.g. yardstick::sensitivity(), yardstick::accuracy(), yardstick::recall(), etc.) and not class probabilities. A set of default metrics is used when NULL (see Details below). |
| na_rm | A single logical: should missing data be removed? |
| event_level | A single string. Either "first" or "second" to specify which level of truth to consider as the "event". |

## Details

Note that that the global option `yardstick.event_first` will be used to determine which level is the event of interest. For more details, see the Relevant level section of `yardstick::sens()`.

The default calculated metrics are:

- `yardstick::j_index()`
- `yardstick::sens()`
- `yardstick::spec()`
- `distance = (1 - sens) ^ 2 + (1 - spec) ^ 2`

If a custom metric is passed that does not compute sensitivity and specificity, the distance metric is not computed.

## Value

A tibble with columns: `.threshold`, `.estimator`, `.metric`, `.estimate` and any existing groups.

## Examples

```
library(dplyr)
data("segment_logistic")

# Set the threshold to 0.6
# > 0.6 = good
# < 0.6 = poor
threshold_perf(segment_logistic, Class, .pred_good, thresholds = 0.6)

# Set the threshold to multiple values
thresholds <- seq(0.5, 0.9, by = 0.1)

segment_logistic %>%
  threshold_perf(Class, .pred_good, thresholds)

# ------------------------------------------------------------------------------

# It works with grouped data frames as well
# Let's mock some resampled data
resamples <- 5

mock_resamples <- resamples %>%
  replicate(
    expr = sample_n(segment_logistic, 100, replace = TRUE),
    simplify = FALSE
  ) %>%
  bind_rows(.id = "resample")

resampled_threshold_perf <- mock_resamples %>%
  group_by(resample) %>%
  threshold_perf(Class, .pred_good, thresholds)
```

```
resampled_threshold_perf

# Average over the resamples
resampled_threshold_perf %>%
  group_by(.metric, .threshold) %>%
  summarise(.estimate = mean(.estimate))
```

# Index