Package 'tailor'

August 25, 2025

Title Iterative Steps for Postprocessing Model Predictions

Version 0.1.0

Description Postprocessors refine predictions outputted from machine learning models to improve predictive performance or better satisfy distributional limitations. This package introduces 'tailor' objects, which compose iterative adjustments to model predictions. A number of pre-written adjustments are provided with the package, such as calibration. See Lichtenstein, Fischhoff, and Phillips (1977) <doi:10.1007/978-94-010-1276-8_19>. Other methods and utilities to compose new adjustments are also included. Tailors are tightly integrated with the 'tidymodels' framework.

License MIT + file LICENSE

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

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

Depends R (>= 4.1)

Imports cli, dplyr, generics, hardhat, purrr, rlang (>= 1.1.0), tibble, tidyselect, vctrs

Suggests betacal, dials (>= 1.4.1), mgcv, modeldata, probably (>= 1.1.0), testthat (>= 3.0.0)

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-04-29

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Simon Couch [aut], Hannah Frick [aut], Emil HvitFeldt [aut], Max Kuhn [aut, cre] (ORCID: <https://orcid.org/0000-0003-2402-136X>), Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Max Kuhn <max@posit.co>

Repository CRAN

Date/Publication 2025-08-25 07:50:03 UTC

Contents

`adjust_equivocal_zone` *Apply an equivocal zone to a binary classification model.*

Description

Equivocal zones describe intervals of predicted probabilities that are deemed too uncertain or ambiguous to be assigned a hard class. Rather than predicting a hard class when the probability is very close to a threshold, tailors using this adjustment predict `"[EQ]"`.

Usage

```
adjust_equivocal_zone(x, value = 0.1, threshold = NULL)
```

Arguments

`x`	A `tailor()`.
`value`	A numeric value (between zero and 1/2) or `hardhat::tune()`. The value is the size of the buffer around the threshold.
`threshold`	A numeric value (between zero and one) or `hardhat::tune()`. Defaults to `adjust_probability_threshold(threshold)` if previously set in `x`, or `1 / 2` if not.

Details

This function transforms the class prediction column `estimate` to have type `class_pred` from `probably::class_pred()`. You can loosely think of this column type as a factor, except there's a possible entry `"[EQ]"` that is *not* a level and will be excluded from performance metric calculations. As a result, the output column has the same number of levels as the input, except now has a possible entry `"[EQ]"` that tidymodels functions know to exclude from further analyses.

Value

An updated `tailor()` containing the new operation.

Data Usage

This adjustment doesn't require estimation and, as such, the same data that's used to train it with `fit()` can be predicted on with `predict()`; fitting this adjustment just collects metadata on the supplied column names and does not risk data leakage.

Examples

```
library(dplyr)
library(modeldata)

head(two_class_example)

# `predicted` gives hard class predictions based on probabilities
two_class_example |> count(predicted)

# when probabilities are within (.25, .75), consider them equivocal
tlr <-
  tailor() |>
  adjust_equivocal_zone(value = 1 / 4)

tlr

# fit by supplying column names.
tlr_fit <- fit(
  tlr,
  two_class_example,
  outcome = c(truth),
  estimate = c(predicted),
  probabilities = c(Class1, Class2)
)

tlr_fit

# adjust hard class predictions
predict(tlr_fit, two_class_example) |> count(predicted)
```

---

adjust_numeric_calibration

Re-calibrate numeric predictions

Description

Calibration for regression models involves adjusting the model's predictions to adjust for correlated errors, ensuring that predicted values align closely with actual observed values across the entire range of outputs.

Usage

```
adjust_numeric_calibration(x, method = NULL, ...)
```

Arguments

x	A tailor().
method	Character. One of "linear", "isotonic","isotonic_boot", or "none", corresponding to the function from the **probably** package probably::cal_estimate_linear(), probably::cal_estimate_isotonic(), or probably::cal_estimate_isotonic_boot(), respectively. The default is to use "linear" which, despite its name, fits a generalized additive model. Note that when fit.tailor() is called, the value may be changed to "none" if there is insufficient data.
...	Optional arguments to pass to the corresponding function in the **probably** package. These arguments must be named.

Details

The "linear" method fits a model that predicts the observed versus the predicted outcome values. This model is used to remove any overt systematic trends from the data, equivalent to removing the model residuals from new data. The underlying code fits that model using mgcv::gam(). If smooth = FALSE is passed to the ..., it uses stats::lm().

The isotonic method uses stats::isoreg() to force the predicted values to increase with the observed outcome. This creates a step function that will map new predictions to values that are monotonically increasing with the outcome. One side effect is that there are fewer, perhaps far fewer, unique predicted values. The "isotonic boot" method resamples the data and generates multiple isotonic regressions that are averaged and used to correct the predictions. This may not be perfectly monotonic, but the number of unique calibrated predictions increases with the number of bootstrap samples (controlled by passing the times argument to ...).

Value

An updated tailor() containing the new operation.

Data Usage

This adjustment requires estimation and, as such, different subsets of data should be used to train it and evaluate its predictions.

Note that, when calling fit.tailor(), if the calibration data have zero or one row, the method is changed to "none".

Examples

```
library(tibble)

# create example data
set.seed(1)
d_calibration <- tibble(y = rnorm(100), y_pred = y/2 + rnorm(100))
d_test <- tibble(y = rnorm(100), y_pred = y/2 + rnorm(100))
```

```
d_calibration

# specify calibration
tlr <-
  tailor() |>
  adjust_numeric_calibration(method = "linear")

# train tailor on a subset of data.
tlr_fit <- fit(tlr, d_calibration, outcome = y, estimate = y_pred)

# apply to predictions on another subset of data
d_test

predict(tlr_fit, d_test)
```

`adjust_numeric_range` *Truncate the range of numeric predictions*

Description

Truncating ranges involves limiting the output of a model to a specific range of values, typically to avoid extreme or unrealistic predictions. This technique can help improve the practical applicability of a model's outputs by constraining them within reasonable bounds based on domain knowledge or physical limitations.

Usage

```
adjust_numeric_range(x, lower_limit = -Inf, upper_limit = Inf)
```

Arguments

`x`	A `tailor()`.
`upper_limit, lower_limit`	A numeric value, NA (for no truncation) or `hardhat::tune()`.

Value

An updated `tailor()` containing the new operation.

Data Usage

This adjustment doesn't require estimation and, as such, the same data that's used to train it with `fit()` can be predicted on with `predict()`; fitting this adjustment just collects metadata on the supplied column names and does not risk data leakage.

Examples

```
library(tibble)

# create example data
set.seed(1)
d <- tibble(y = rnorm(100), y_pred = y/2 + rnorm(100))
d

# specify calibration
tlr <-
  tailor() |>
  adjust_numeric_range(lower_limit = 1)

# train tailor by passing column names.
tlr_fit <- fit(tlr, d, outcome = y, estimate = y_pred)

predict(tlr_fit, d)
```

---

adjust_predictions_custom

Change or add variables

Description

This adjustment functions allows for arbitrary transformations of model predictions using `dplyr::mutate()` statements.

Usage

```
adjust_predictions_custom(x, ..., .pkgs = character(0))
```

Arguments

`x`	A `tailor()`.
`...`	Name-value pairs of expressions. See `dplyr::mutate()`.
`.pkgs`	A character string of extra packages that are needed to execute the commands.

Value

An updated `tailor()` containing the new operation.

Data-dependent transformations

Note that custom adjustments should not carry out estimation. If they do, the estimation steps will be carried out independently at `fit()` and `predict()` time. For example, if your transformation includes a mean shift, the postprocessor will take the mean of the column supplied in the training data at `fit()` and, rather than reusing that mean at `predict()` will take the mean again of the dataset supplied at `predict()` time.

Data Usage

This adjustment doesn't require estimation and, as such, the same data that's used to train it with `fit()` can be predicted on with `predict()`; fitting this adjustment just collects metadata on the supplied column names and does not risk data leakage.

Examples

```
library(modeldata)

head(two_class_example)

tlr <-
  tailor() |>
  adjust_equivocal_zone() |>
  adjust_predictions_custom(linear_predictor = binomial()$linkfun(Class2))

tlr_fit <- fit(
  tlr,
  two_class_example,
  outcome = c(truth),
  estimate = c(predicted),
  probabilities = c(Class1, Class2)
)

predict(tlr_fit, two_class_example) |> head()
```

`adjust_probability_calibration` *Re-calibrate classification probability predictions*

Description

Calibration is the process of adjusting a model's outputted probabilities to match the observed frequencies of events. This technique aims to ensure that when a model predicts a certain probability for an outcome, that probability accurately reflects the true likelihood of that outcome occurring.

Usage

```
adjust_probability_calibration(x, method = NULL, ...)
```

Arguments

`x`	A `tailor()`.
`method`	Character. One of `"logistic"`, `"multinomial"`, `"beta"`, `"isotonic"`, `"isotonic_boot"`, or `"none"`, corresponding to the function from the **probably** package `probably::cal_estimate_logist` `probably::cal_estimate_multinomial()`, etc., respectively. The default is to use `"logistic"` which, despite its name, fits a generalized additive model.

Note that when `fit.tailor()` is called, the value may be changed to `"none"` if there is insufficient data.

`...` Optional arguments to pass to the corresponding function in the **probably** package. These arguments must be named.

Details

The "logistic" and "multinomial" methods fit models that predict the observed classes as a function of the predicted class probabilities. These models remove any overt systematic trends from the linear predictor and correct new predictions. The underlying code fits that model using `mgcv::gam()`. If `smooth = FALSE` is passed to the `...`, it uses `stats::glm()` for binary outcomes or `nnet::multinom()` for 3+ classes.

The isotonic method uses `stats::isoreg()` to force the predicted probabilities to increase with the observed outcome class. This creates a step function that will map new predictions to values that are monotonically increasing with the binary (0/1) form of the outcome. One side effect is that there are fewer, perhaps far fewer, unique predicted probabilities. For 3+ classes, this is done using a one-versus-all strategy that ensures that the probabilities add to 1.0. The "isotonic boot" method resamples the data and generates multiple isotonic regressions that are averaged and used to correct the predictions. This may not be perfectly monotonic, but the number of unique calibrated predictions increases with the number of bootstrap samples (controlled by passing the `times` argument to `...`).

Beta calibration (Kull *et al*, 2017) assumes that the probability estimates follow a Beta distribution. This leads to a sigmoidal model that can be fit to the data via maximum likelihood. There are a few different ways to fit the model; see `betacal:: beta_calibration()` options `parameters` to select a specific sigmoidal model.

Value

An updated `tailor()` containing the new operation.

Data Usage

This adjustment requires estimation and, as such, different subsets of data should be used to train it and evaluate its predictions.

Note that, when calling `fit.tailor()`, if the calibration data have zero or one row, the `method` is changed to `"none"`.

References

Kull, Meelis, Telmo Silva Filho, and Peter Flach. "Beta calibration: a well-founded and easily implemented improvement on logistic calibration for binary classifiers." Artificial intelligence and statistics. PMLR, 2017.

https://aml4td.org/chapters/cls-metrics.html#sec-cls-calibration

Examples

```
library(modeldata)
```

```
# split example data
set.seed(1)
in_rows <- sample(c(TRUE, FALSE), nrow(two_class_example), replace = TRUE)
d_calibration <- two_class_example[in_rows, ]
d_test <- two_class_example[!in_rows, ]

head(d_calibration)

# specify calibration
tlr <-
  tailor() |>
  adjust_probability_calibration(method = "logistic")

# train tailor on a subset of data.
tlr_fit <- fit(
  tlr,
  d_calibration,
  outcome = c(truth),
  estimate = c(predicted),
  probabilities = c(Class1, Class2)
)

# apply to predictions on another subset of data
head(d_test)

predict(tlr_fit, d_test)
```

`adjust_probability_threshold` *Change the event threshold*

Description

Many machine learning systems determine hard class predictions by first predicting the probability of an event and then predicting that an event will occur if its respective probability is above 0.5. This adjustment allows practitioners to determine hard class predictions using a threshold other than 0.5. By setting appropriate thresholds, one can balance the trade-off between different types of errors (such as false positives and false negatives) to optimize the model's performance for specific use cases.

Usage

```
adjust_probability_threshold(x, threshold = 0.5)
```

Arguments

`x`	A `tailor()`.
`threshold`	A numeric value (between zero and one) or `hardhat::tune()`.

Value

An updated `tailor()` containing the new operation.

Data Usage

This adjustment doesn't require estimation and, as such, the same data that's used to train it with `fit()` can be predicted on with `predict()`; fitting this adjustment just collects metadata on the supplied column names and does not risk data leakage.

Examples

```
library(modeldata)

# `predicted` gives hard class predictions based on probability threshold .5
head(two_class_example)

# use a threshold of .1 instead:
tlr <-
  tailor() |>
  adjust_probability_threshold(.1)

# fit by supplying column names.
tlr_fit <- fit(
  tlr,
  two_class_example,
  outcome = c(truth),
  estimate = c(predicted),
  probabilities = c(Class1, Class2)
)

# adjust hard class predictions
predict(tlr_fit, two_class_example) |> head()
```

`fit.tailor` *Fit and predict from tailors*

Description

These functions apply `fit()` and `predict()` methods for each adjustment added to a tailor, in the order in which they were applied.

Usage

```
## S3 method for class 'tailor'
fit(object, .data, outcome, estimate, probabilities = c(), ...)

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

Arguments

object	A tailor().
.data, new_data	A data frame containing predictions from a model.
outcome	<tidy-select> The column name of the outcome variable.
estimate	<tidy-select>
probabilities	<tidy-select> The column names of class probability estimates. These should be given in the order of the factor levels of the estimate.
...	Currently ignored.

Value

An updated tailor() objects. Any estimates produced and saved by fit.tailor() are saved in the adjustments element of the tailor.

Data Usage

For adjustments that don't require estimating parameters, training with fit() simply evaluates tidyselect expressions and logs column names. For others, as in adjust_numeric_calibration(), adjustments actually learn from data; in that case, separate subsets of data ought to be used for training the tailor and evaluating its performance on predictions.

Note that if .data has zero or one row, the method is changed to "none".

Examples

```
library(modeldata)

# `predicted` gives hard class predictions based on probability threshold .5
head(two_class_example)

# use a threshold of .1 instead:
tlr <-
  tailor() |>
  adjust_probability_threshold(.1)

# fit by supplying column names.
tlr_fit <- fit(
  tlr,
  two_class_example,
  outcome = c(truth),
  estimate = c(predicted),
  probabilities = c(Class1, Class2)
)

# adjust hard class predictions
predict(tlr_fit, two_class_example) |> head()
```

`tailor` *Declare post-processing for model predictions*

Description

Tailors compose iterative adjustments to model predictions. After initializing a tailor with this function, add adjustment specifications with `adjust_*()` functions:

- For probability distributions: `adjust_probability_calibration()`
- For transformation of probabilities to hard class predictions: `adjust_probability_threshold()`, `adjust_equivocal_zone()`
- For numeric distributions: `adjust_numeric_calibration()`, `adjust_numeric_range()`

For ad-hoc adjustments, see `adjust_predictions_custom()`.

Tailors must be trained with fit() before being applied to new data with predict().

Usage

```
tailor()
```

Value

An object of class `tailor` with elements:

- `type`: The type of task (e.g., regression)
- `adjustments`: A list containing the sequential options specified by the user.
- `columns`: the data set column names for the true outcome values and the predictions of various types. If these are not specified, then `NULL`. These are usual
- `ptype`: a zero-row slice of the data containing the `columns`.

Most of these values are set when an adjustment is added to the tailor or when `fit.tailor()` is used.

Ordering of adjustments

When composing multiple adjustments in a tailor object, the order matters and must follow specific rules depending on the type of predictions being adjusted (classification or regression).

For classification problems (binary and multiclass), adjustments that modify probability estimates (e.g., `adjust_probability_calibration()`) must be applied before adjustments that change hard class predictions (including `adjust_equivocal_zone()`). This ensures that class predictions are based on the final calibrated probabilities. For regression problems, `adjust_numeric_calibration()` must be applied before other numeric adjustments. This ensures that subsequent adjustments work with calibrated predictions.

Generally, adjustments cannot be duplicated (i.e. the same adjustment type cannot be used multiple times in a tailor object), though `adjust_predictions_custom()` can be used multiple times. Adjustments for different prediction types cannot be mixed—numeric adjustments (for regression) and probability adjustments (for classification) cannot be used in the same tailor object.

If these ordering rules are violated, `tailor()` will raise an error describing the issue.

Examples

```
library(dplyr)
library(modeldata)

# `predicted` gives hard class predictions based on probabilities
two_class_example |> count(predicted)

# change the probability threshold to allot one class vs the other
tlr <-
  tailor() |>
  adjust_probability_threshold(threshold = .1)

tlr

# fit by supplying column names.
tlr_fit <- fit(
  tlr,
  two_class_example,
  outcome = c(truth),
  estimate = c(predicted),
  probabilities = c(Class1, Class2)
)

tlr_fit

# adjust hard class predictions
predict(tlr_fit, two_class_example) |> count(predicted)
```

`tidy.tailor` *Tidy a tailor object*

Description

Describe a tailor's adjustments in a tibble with one row per adjustment.

Usage

```
## S3 method for class 'tailor'
tidy(x, number = NA, ...)
```

Arguments

`x`	A `tailor()` object.
`number`	Optional. A single integer between 1 and the number of adjustments.
`...`	Currently unused; must be empty.

Value

A tibble containing information about the tailor's adjustments including their ordering, whether they've been trained, and whether they require training with a separate calibration set.

Examples

```
tailor() |>
  adjust_numeric_range(lower_limit = 1) |>
  tidy()
```

Index