Package 'tidyposterior'

July 31, 2025

Title Bayesian Analysis to Compare Models using Resampling Statistics

Version 1.0.2

Description Bayesian analysis used here to answer the question: ``when looking at resampling results, are the differences between models 'real'?" To answer this, a model can be created were the performance statistic is the resampling statistics (e.g. accuracy or RMSE). These values are explained by the model types. In doing this, we can get parameter estimates for each model's affect on performance and make statistical (and practical) comparisons between models. The methods included here are similar to Benavoli et al (2017) <https://jmlr.org/papers/v18/16-305.html>.

License MIT + file LICENSE

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

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

Depends R (>= 4.1)

Imports dplyr (> 1.0.0), generics, ggplot2 (>= 3.5.2), purrr, rlang, rsample (>= 0.0.2), rstanarm (>= 2.21.1), stats, tibble, tidyr (>= 0.7.1), tune (>= 0.2.0), utils, vctrs (>= 0.3.0), workflowsets

Suggests covr, knitr, parsnip, rmarkdown, splines2, testthat (>= 3.0.0), yardstick

VignetteBuilder knitr

ByteCompile true

Config/Needs/website tidymodels, tidyverse/tidytemplate

Config/testthat/edition 3

Config/usethis/last-upkeep 2025-04-27

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

NeedsCompilation no

Author 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 2025-07-31 12:00:02 UTC

Contents

autoplot.posterior 2
autoplot.posterior_diff 3
contrast_models 4
no_trans 4
perf_mod 5
precise_example 11
summary.posterior 12
summary.posterior_diff 13
tidy.perf_mod 14

Index **15**

`autoplot.posterior` *Visualize the Posterior Distributions of Model Statistics*

Description

For objects of classes `posterior` and `perf_mod`, `autoplot()` produces a simple plot of posterior distributions. For workflow set objects, there are several types of plots that can be produced.

Usage

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

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

## S3 method for class 'perf_mod_workflow_set'
autoplot(object, type = "intervals", prob = 0.9, size = NULL, ...)
```

Arguments

`object`	An object produced by `perf_mod()`, `tidy.perf_mod()`, or a workflow set with computed results.
`...`	Options passed to `geom_line(stat = "density", ...)`.

type	A value of one of: "intervals" (for model rank versus posterior probability using interval estimation), "posteriors" (density plots for each model), or "ROPE" (for practical equivalence probabilities versus workflow rank).
prob	A number p (0 < p < 1) indicating the desired probability mass to include in the intervals.
size	The size of an effective difference in the units of the chosen metric. For example, a 5 percent increase in accuracy (size = 0.05) between two models might be considered a "real" difference.

Value

A ggplot2::ggplot() object.

Examples

```
data(ex_objects)
autoplot(posterior_samples)
```

---

autoplot.posterior_diff

Visualize the Posterior Distributions of Model Differences

Description

A density is created for each contrast in a faceted grid.

Usage

```
## S3 method for class 'posterior_diff'
autoplot(object, size = 0, ...)
```

Arguments

object	An object produced by contrast_models().
size	The size of an effective difference. For example, a 5\ "real" difference.
...	Options passed to geom_line(stat = "density", ...).

Value

A ggplot2::ggplot() object using geom_density faceted by the models being contrasted (when there are 2 or more contrasts).

Examples

```
data(ex_objects)
library(ggplot2)
autoplot(contrast_samples)
```

contrast_models *Estimate the Difference Between Models*

Description

The posterior distributions created by `perf_mod()` can be used to obtain the posterior distribution of the difference(s) between models. One or more comparisons can be computed at the same time.

Usage

```
contrast_models(x, list_1 = NULL, list_2 = NULL, seed = sample.int(10000, 1))
```

Arguments

`x`	An object produced by `perf_mod()`.
`list_1, list_2`	Character vectors of equal length that specify the specific pairwise contrasts. The contrast is parameterized as `list_1[i] - list_2[i]`. If the defaults are left to `NULL`, all combinations are evaluated.
`seed`	A single integer for sampling from the posterior.

Details

If a transformation was used when `x` was created, the inverse is applied *before* the difference is computed.

Value

A data frame of the posterior distribution(s) of the difference(s). The object has an extra class of `"posterior_diff"`.

no_trans *Simple Transformation Functions*

Description

A set of objects are contained here to easily facilitate the use of outcome transformations for modeling. For example, if there is a large amount of variability in the resampling results for the Kappa statistics, which lies between -1 and 1, assuming normality may produce posterior estimates outside of the natural bound. One way to solve this is to use a link function or assume a prior that is appropriately bounded. Another approach is to transform the outcome values prior to modeling using a Gaussian prior and reverse-transforming the posterior estimates prior to visualization and summarization. These object can help facilitate this last approach.

Usage

```
no_trans

logit_trans

Fisher_trans

ln_trans

inv_trans
```

Format

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

An object of class `list` of length 2.

Details

The `logit_trans` object is useful for model performance statistics bounds in zero and one, such as accuracy or the area under the ROC curve.

`ln_trans` and `inv_trans` can be useful when the statistics are right-skewed and strictly positive.

`Fisher_trans` was originally used for correlation statistics but can be used here for an metrics falling between -1 and 1, such as Kappa.

Examples

```
logit_trans$func(.5)
logit_trans$inv(0)
```

---

perf_mod *Bayesian Analysis of Resampling Statistics*

Description

Bayesian analysis used here to answer the question: "when looking at resampling results, are the differences between models 'real?'" To answer this, a model can be created were the *outcome* is the resampling statistics (e.g. accuracy or RMSE). These values are explained by the model types. In doing this, we can get parameter estimates for each model's affect on performance and make statistical (and practical) comparisons between models.

Usage

```
perf_mod(object, ...)

## S3 method for class 'rset'
perf_mod(object, transform = no_trans, hetero_var = FALSE, formula = NULL, ...)

## S3 method for class 'resamples'
perf_mod(
  object,
  transform = no_trans,
  hetero_var = FALSE,
  metric = object$metrics[1],
  ...
)

## S3 method for class 'data.frame'
perf_mod(object, transform = no_trans, hetero_var = FALSE, formula = NULL, ...)

## S3 method for class 'tune_results'
perf_mod(
  object,
  metric = NULL,
  transform = no_trans,
  hetero_var = FALSE,
  formula = NULL,
  filter = NULL,
  ...
)

## S3 method for class 'workflow_set'
perf_mod(
  object,
  metric = NULL,
  transform = no_trans,
  hetero_var = FALSE,
  formula = NULL,
  ...
)
```

Arguments

`object`	Depending on the context (see Details below): • A data frame with `id` columns for the resampling groupds and metric results in all of the other columns.. • An `rset` object (such as `rsample::vfold_cv()`) containing the `id` column(s) and at least two numeric columns of model performance statistics (e.g. accuracy). • An object from `caret::resamples`.

- An object with class `tune_results`, which could be produced by `tune::tune_grid()`, `tune::tune_bayes()` or similar.
- A workflow set where all results contain the metric value given in the `metric` argument value.

`...` Additional arguments to pass to `rstanarm::stan_glmer()` such as `verbose`, `prior`, `seed`, `refresh`, `family`, etc.

`transform` An named list of transformation and inverse transformation functions. See `logit_trans()` as an example.

`hetero_var` A logical; if `TRUE`, then different variances are estimated for each model group. Otherwise, the same variance is used for each group. Estimating heterogeneous variances may slow or prevent convergence.

`formula` An optional model formula to use for the Bayesian hierarchical model (see Details below).

`metric` A single character value for the statistic from the `resamples` object that should be analyzed.

`filter` A conditional logic statement that can be used to filter the statistics generated by `tune_results` using the tuning parameter values or the `.config` column.

Details

These functions can be used to process and analyze matched resampling statistics from different models using a Bayesian generalized linear model with effects for the model and the resamples.

Bayesian Model formula:

By default, a generalized linear model with Gaussian error and an identity link is fit to the data and has terms for the predictive model grouping variable. In this way, the performance metrics can be compared between models.

Additionally, random effect terms are also used. For most resampling methods (except repeated V-fold cross-validation), a simple random intercept model its used with an exchangeable (i.e. compound-symmetric) variance structure. In the case of repeated cross-validation, two random intercept terms are used; one for the repeat and another for the fold within repeat. These also have exchangeable correlation structures.

The above model specification assumes that the variance in the performance metrics is the same across models. However, this is unlikely to be true in some cases. For example, for simple binomial accuracy, it well know that the variance is highest when the accuracy is near 50 percent. When the argument `hetero_var = TRUE`, the variance structure uses random intercepts for each model term. This may produce more realistic posterior distributions but may take more time to converge.

Examples of the default formulas are:

```
# One ID field and common variance:
  statistic ~ model + (model | id)

# One ID field and heterogeneous variance:
  statistic ~ model + (model + 0 | id)

# Repeated CV (id = repeat, id2 = fold within repeat)
```

```
# with a common variance:
  statistic ~ model + (model | id2/id)

# Repeated CV (id = repeat, id2 = fold within repeat)
# with a heterogeneous variance:
  statistic ~ model + (model + 0| id2/id)

# Default for unknown resampling method and
# multiple ID fields:
  statistic ~ model + (model | idN/../id)
```

Custom formulas should use `statistic` as the outcome variable and `model` as the factor variable with the model names.

Also, as shown in the package vignettes, the Gaussian assumption make be unrealistic. In this case, there are at least two approaches that can be used. First, the outcome statistics can be transformed prior to fitting the model. For example, for accuracy, the logit transformation can be used to convert the outcome values to be on the real line and a model is fit to these data. Once the posterior distributions are computed, the inverse transformation can be used to put them back into the original units. The `transform` argument can be used to do this.

The second approach would be to use a different error distribution from the exponential family. For RMSE values, the Gamma distribution may produce better results at the expense of model computational complexity. This can be achieved by passing the `family` argument to `perf_mod` as one might with the `glm` function.

Input formats:

There are several ways to give resampling results to the `perf_mod()` function. To illustrate, here are some example objects using 10-fold cross-validation for a simple two-class problem:

```
library(tidymodels)
library(tidyposterior)
library(workflowsets)

data(two_class_dat, package = "modeldata")

set.seed(100)
folds <- vfold_cv(two_class_dat)
```

We can define two different models (for simplicity, with no tuning parameters).

```
logistic_reg_glm_spec <-
  logistic_reg() |>
  set_engine('glm')

mars_earth_spec <-
  mars(prod_degree = 1) |>
  set_engine('earth') |>
  set_mode('classification')
```

For tidymodels, the `tune::fit_resamples()` function can be used to estimate performance for each model/resample:

```
rs_ctrl <- control_resamples(save_workflow = TRUE)

logistic_reg_glm_res <-
  logistic_reg_glm_spec |>
  fit_resamples(Class ~ ., resamples = folds, control = rs_ctrl)

mars_earth_res <-
  mars_earth_spec |>
  fit_resamples(Class ~ ., resamples = folds, control = rs_ctrl)
```

From these, there are several ways to pass the results to `perf_mod()`.

Data Frame as Input:
The most general approach is to have a data frame with the resampling labels (i.e., one or more id columns) as well as columns for each model that you would like to compare.
For the model results above, `tune::collect_metrics()` can be used along with some basic data manipulation steps:

```
logistic_roc <-
  collect_metrics(logistic_reg_glm_res, summarize = FALSE) |>
  dplyr::filter(.metric == "roc_auc") |>
  dplyr::select(id, logistic = .estimate)

mars_roc <-
  collect_metrics(mars_earth_res, summarize = FALSE) |>
  dplyr::filter(.metric == "roc_auc") |>
  dplyr::select(id, mars = .estimate)

resamples_df <- full_join(logistic_roc, mars_roc, by = "id")
resamples_df
```

```
## # A tibble: 10 x 3
##   id     logistic  mars
##   <chr>     <dbl> <dbl>
## 1 Fold01    0.908 0.875
## 2 Fold02    0.904 0.917
## 3 Fold03    0.924 0.938
## 4 Fold04    0.881 0.881
## 5 Fold05    0.863 0.864
## 6 Fold06    0.893 0.889
## # . . .  with 4 more rows
```

We can then give this directly to `perf_mod()`:

```
set.seed(101)
roc_model_via_df <- perf_mod(resamples_df, refresh = 0)
tidy(roc_model_via_df) |> summary()
```

```
## # A tibble: 2 x 4
##   model     mean lower upper
##   <chr>    <dbl> <dbl> <dbl>
## 1 logistic 0.892 0.879 0.906
## 2 mars     0.888 0.875 0.902
```

rsample Object as Input:
Alternatively, the result columns can be merged back into the original `rsample` object. The up-side to using this method is that `perf_mod()` will know exactly which model formula to use for the Bayesian model:

```
resamples_rset <-
  full_join(folds, logistic_roc, by = "id") |>
  full_join(mars_roc, by = "id")

set.seed(101)
roc_model_via_rset <- perf_mod(resamples_rset, refresh = 0)
tidy(roc_model_via_rset) |> summary()
```

```
## # A tibble: 2 x 4
##   model     mean lower upper
##   <chr>    <dbl> <dbl> <dbl>
## 1 logistic 0.892 0.879 0.906
## 2 mars     0.888 0.875 0.902
```

Workflow Set Object as Input:
Finally, for tidymodels, a workflow set object can be used. This is a collection of models/preprocessing combinations in one object. We can emulate a workflow set using the existing example results then pass that to `perf_mod()`:

```
example_wset <-
  as_workflow_set(logistic = logistic_reg_glm_res, mars = mars_earth_res)

set.seed(101)
roc_model_via_wflowset <- perf_mod(example_wset, refresh = 0)
tidy(roc_model_via_rset) |> summary()
```

```
## # A tibble: 2 x 4
##   model     mean lower upper
##   <chr>    <dbl> <dbl> <dbl>
## 1 logistic 0.892 0.879 0.906
## 2 mars     0.888 0.875 0.902
```

caret resamples object:
The `caret` package can also be used. An equivalent set of models are created:

```
library(caret)

set.seed(102)
logistic_caret <- train(Class ~ ., data = two_class_dat, method = "glm",
                        trControl = trainControl(method = "cv"))

set.seed(102)
mars_caret <- train(Class ~ ., data = two_class_dat, method = "gcvEarth",
                    tuneGrid = data.frame(degree = 1),
                    trControl = trainControl(method = "cv"))
```

Note that these two models use the same resamples as one another due to setting the seed prior to calling `train()`. However, these are different from the tidymodels results used above (so the final results will be different).

caret has a resamples() function that can collect and collate the resamples. This can also be given to perf_mod():

```
caret_resamples <- resamples(list(logistic = logistic_caret, mars = mars_caret))

set.seed(101)
roc_model_via_caret <- perf_mod(caret_resamples, refresh = 0)
tidy(roc_model_via_caret) |> summary()
## # A tibble: 2 x 4
##   model     mean lower upper
##   <chr>    <dbl> <dbl> <dbl>
## 1 logistic 0.821 0.801 0.842
## 2 mars     0.822 0.802 0.842
```

Value

An object of class perf_mod. If a workfkow set is given in object, there is an extra class of "perf_mod_workflow_set".

References

Kuhn and Silge (2021) *Tidy Models with R*, Chapter 11, https://www.tmwr.org/compare.html

See Also

tidy.perf_mod(), contrast_models()

---

precise_example *Example Data Sets*

Description

Example Data Sets

Details

Several data sets are contained in the package as examples. Each *simulates* an rset object but the splits columns are not included to save space.

- precise_example contains the results of the classification analysis of a real data set using 10-fold CV. The holdout data sets contained thousands of examples and have precise performance estimates. Three models were fit to the original data and several performance metrics are included.
- noisy_example was also generated from a regression data simulation. The original data set was small (50 samples) and 10-repeated of 10-fold CV were used with four models. There is an excessive of variability in the results (probably more than the resample-to-resample variability). The RMSE distributions show fairly right-skewed distributions.

- `concrete_example` contains the results of the regression case study from the book *Applied Predictive Modeling*. The original data set contained 745 samples in the training set. 10-repeats of 10-fold CV was also used and 13 models were fit to the data.
- `ts_example` is from a data set where rolling-origin forecast resampling was used. Each assessment set is the summary of 14 observations (i.e. 2 weeks). The analysis set consisted of a base of about 5,500 samples plus the previous assessment sets. Four regression models were applied to these data.
- `ex_object` objects were generated from the `two_class_dat` data in the `modeldata` package. Basic 10-fold cross validation was used to evaluate the models. The `posterior_samples` object is samples of the posterior distribution of the model ROC values while `contrast_samples` are posterior probabilities form the differences in ROC values.

Value

Tibbles with the additional class `rset`

Examples

```
data(precise_example)
precise_example
```

`summary.posterior` *Summarize the Posterior Distributions of Model Statistics*

Description

Numerical summaries are created for each model including the posterior mean and upper and lower credible intervals (aka uncertainty intervals).

Usage

```
## S3 method for class 'posterior'
summary(object, prob = 0.9, seed = sample.int(10000, 1), ...)
```

Arguments

`object`	An object produced by `tidy.perf_mod()`.
`prob`	A number p ($0 < p < 1$) indicating the desired probability mass to include in the intervals.
`seed`	A single integer for sampling from the posterior.
`...`	Not currently used

Value

A data frame with summary statistics and a row for each model.

Examples

```
data("ex_objects")

summary(posterior_samples)
```

`summary.posterior_diff` *Summarize Posterior Distributions of Model Differences*

Description

Credible intervals are created for the differences. Also, region of practical equivalence (ROPE) statistics are computed when the effective size of a difference is given.

Usage

```
## S3 method for class 'posterior_diff'
summary(object, prob = 0.9, size = 0, ...)
```

Arguments

`object`	An object produced by `contrast_models()`.
`prob`	A number p (0 < p < 1) indicating the desired probability mass to include in the intervals.
`size`	The size of an effective difference in the units of the chosen metric. For example, a 5 percent increase in accuracy (`size = 0.05`) between two models might be considered a "real" difference.
`...`	Not currently used

Details

The ROPE estimates included in the results are the columns `pract_neg`, `pract_equiv`, and `pract_pos`. `pract_neg` integrates the portion of the posterior below `-size` (and `pract_pos` is the upper integral starting at `size`). The interpretation depends on whether the metric being analyzed is better when larger or smaller. `pract_equiv` integrates between `[-size, size]`. If this is close to one, the two models are unlikely to be practically different relative to `size`.

Value

A data frame with interval and ROPE statistics for each comparison.

Examples

```
data("ex_objects")

summary(contrast_samples)
summary(contrast_samples, size = 0.025)
```

`tidy.perf_mod` *Extract Posterior Distributions for Models*

Description

`tidy` can be used on an object produced by `perf_mod()` to create a data frame with a column for the model name and the posterior predictive distribution values.

Usage

```
## S3 method for class 'perf_mod'
tidy(x, seed = sample.int(10000, 1), ...)
```

Arguments

`x`	An object from `perf_mod()`
`seed`	A single integer for sampling from the posterior.
`...`	Not currently used

Details

Note that this posterior only reflects the variability of the groups (i.e. the fixed effects). This helps answer the question of which model is best *for this data set*. If does not answer the question of which model would be best on a new resample of the data (which would have greater variability).

Value

A data frame with the additional class `"posterior"`

Index

∗ **datasets**
    no_trans, 4
    precise_example, 11

autoplot.perf_mod *(*autoplot.posterior*)*, 2
autoplot.perf_mod_workflow_set *(*autoplot.posterior*)*, 2
autoplot.posterior, 2
autoplot.posterior_diff, 3

concrete_example *(*precise_example*)*, 11
contrast_models, 4
contrast_models(), *3*, *11*, *13*
contrast_samples *(*precise_example*)*, 11

ex_object *(*precise_example*)*, 11

Fisher_trans *(*no_trans*)*, 4

ggplot2::ggplot(), *3*

inv_trans *(*no_trans*)*, 4

ln_trans *(*no_trans*)*, 4
logit_trans *(*no_trans*)*, 4
logit_trans(), *7*

no_trans, 4
noisy_example *(*precise_example*)*, 11

perf_mod, 5
perf_mod(), *2*, *4*, *14*
posterior_samples *(*precise_example*)*, 11
precise_example, 11

rsample::vfold_cv(), *6*
rstanarm::stan_glmer(), *7*

summary.posterior, 12
summary.posterior_diff, 13

tidy.perf_mod, 14
tidy.perf_mod(), *2*, *11*, *12*
ts_example *(*precise_example*)*, 11
tune::collect_metrics(), *9*
tune::fit_resamples(), *8*