Package ‘HuraultMisc’

October 12, 2022

Title  Guillem Hurault Functions' Library
Version  1.1.1
Description  Contains various functions for data analysis, notably helpers and diagnostics for Bayesian modelling using Stan.
URL  https://github.com/ghurault/HuraultMisc
BugReports  https://github.com/ghurault/HuraultMisc/issues
Depends  R (>= 3.5.0)
License  MIT + file LICENSE
Encoding  UTF-8
LazyData  true
Imports  ggplot2, reshape2, rstan, cowplot, Hmisc, stats, grDevices, HDInterval, magrittr, dplyr, tidyr
RoxygenNote  7.1.1
Suggests  testthat (>= 2.1.0), covr
NeedsCompilation  no
Author  Guillem Hurault [aut, cre] (<https://orcid.org/0000-0002-1052-3564>)
Maintainer  Guillem Hurault <guillem.hurault@hotmail.fr>
Repository  CRAN
Date/Publication  2021-09-06 08:20:16 UTC

R topics documented:

approx_equal ................................................................. 2
cbbPalette ................................................................. 3
change_colnames ............................................................ 3
compute_calibration ......................................................... 4
compute_resolution .......................................................... 5
compute_RPS ................................................................. 5
coverage ................................................................. 6
empirical_pval ............................................................... 7
approx_equal

Description
Compute whether x and y are approximately equal given a tolerance level

Usage
approx_equal(x, y, tol = .Machine$double.eps^0.5)

x %~% y

Arguments
x Numeric scalar.
y Numeric scalar.
tol Tolerance.

Value
Boolean
Examples

approx_equal(1, 1)
1 %-% (1 + 1e-16)
1 %-% 1.01

---

cbbPalette

A colorblind-friendly palette (with black)

Description


Usage

cbbPalette

Format

An object of class character of length 8.

Source

Cookbook for R

---

change_colnames

Change column names of a dataframe

Description

Change column names of a dataframe

Usage

change_colnames(df, current_names, new_names)

Arguments

- df: Dataframe
- current_names: Vector of column names to change.
- new_names: Vector of new names.

Value

Dataframe with new column names
compute_calibration

Estimate calibration given forecasts and corresponding outcomes

df <- data.frame(A = 1:2, B = 3:4, C = 5:6)
df <- change_colnames(df, c("A", "C"), c("Aa", "Cc"))

compute_calibration(forecast, outcome, method = c("smoothing", "binning"), CI = NULL, binwidth = NULL, ...)

Arguments

- forecast: Vector of probability forecasts.
- outcome: Vector of observations (0 or 1).
- method: Method used to estimate calibration, either "smoothing" or "binning".
- CI: Confidence level (e.g. 0.95). CI not computed if NULL (CI can be expensive to compute for LOWESS).
- binwidth: Binwidth when calibration is estimated by binning. If NULL, automatic bin width selection with 'Sturges' method.
- ...: Arguments of stats::loess() function (e.g. span)

Value

Dataframe with columns Forecast (bins), Frequency (frequency of outcomes in the bin), Lower (lower bound of the CI) and Upper (upper bound of the CI).

Examples

N <- 1e4
f <- rbeta(N, 1, 1)
o <- sapply(f, function(x) {rbinom(1, 1, x)})
lapply(c("binning", "smoothing"),
  function(m) {
    cal <- compute_calibration(f, o, method = m)
compute_resolution

```r
with(cal, plot(Forecast, Frequency, type = "l")
abline(c(0, 1), col = "red")
}
```

compute_resolution  
Compute resolution of forecasts, normalised by the uncertainty

Description
The resolution is computed as the mean squared distance to a base rate (reference forecast) and is then normalised by the uncertainty (maximum resolution). This means the output is between 0 and 1, 1 corresponding to the maximum resolution.

Usage
```r
compute_resolution(f, p0)
```

Arguments
- `f`  
  Vector of forecasts
- `p0`  
  Vector of base rate. In the case rate is usually the prevalence of a uniform forecast (e.g. 1 / number of categories) but can depend on the observation (hence the vector).

Value
Vector of resolution values

Examples
```r
compute_resolution(seq(0, 1, .1), 0.5)
```

compute_RPS  
Compute RPS for a single forecast

Description
Compute RPS for a single forecast

Usage
```r
compute_RPS(forecast, outcome)
```

Arguments
- `forecast`  
  Vector of length N (forecast).
- `outcome`  
  Index of the true outcome (between 1 and N).
Value

RPS (numeric scalar)

Examples

compute_RPS(c(.2, .5, .3), 2)

<table>
<thead>
<tr>
<th>coverage</th>
<th>Coverage probability</th>
</tr>
</thead>
</table>

Description

Compute and plot coverage of CI for different confidence level. Useful for fake data check.

Usage

```r
compute_coverage(
    post_samples,
    truth,
    CI = seq(0, 1, 0.05),
    type = c("eti", "hdi")
)
```

```r
plot_coverage(
    post_samples,
    truth,
    CI = seq(0, 1, 0.05),
    type = c("eti", "hdi")
)
```

Arguments

- `post_samples`: Matrix of posterior samples. Rows represent a sample and columns represent variables.
- `truth`: Vector of true parameter values (should be the same length as the number of columns in `post_samples`).
- `CI`: Vector of confidence levels.
- `type`: Type of confidence intervals: either "eti" (equal-tailed intervals) or "hdi" (highest density intervals).

Value

compute_coverage returns a Dataframe containing coverage (and 95% uncertainty interval for the coverage) for different confidence level (nominal coverage). plot_coverage returns a ggplot of the coverage as the function of the nominal coverage with 95% uncertainty interval.
**empirical_pval**

Compute empirical p-values

### Description

Compute empirical p-values

### Usage

```r
empirical_pval(t_rep, t, alternative = c("two.sided", "less", "greater"))
```

### Arguments

- **t_rep**
  - Vector of samples from a distribution.
- **t**
  - Observation (numeric scalar).
- **alternative**
  - Indicates the alternative hypothesis: must be one of "two.sided", "greater" or "less".

### Value

Empirical p-value.

### Examples

```r
empirical_pval(rnorm(1e2), 2)
```
### extract_ci

*Extract confidence intervals from a vector of samples*

**Description**

Extract confidence intervals from a vector of samples

**Usage**

```r
evaluate(x, CI_level = seq(0.1, 0.9, 0.1), type = c("eti", "hdi"))
```

**Arguments**

- **x** Vector of samples from a distribution.
- **CI_level** Vector containing the level of the confidence/credible intervals.
- **type** "eti" for equal-tailed intervals and "hdi" for highest density intervals.

**Value**

Dataframe with columns: Lower, Upper, Level.

**Examples**

```r
x <- rexp(1e4)
evaluate(x, type = "eti")
evaluate(x, type = "hdi")
```

### extract_distribution

*Extract a distribution represented by samples*

**Description**

The distribution can be extracted as:

- a probability density function ("continuous").
- a probability mass function ("discrete").
- a series of equal-tailed confidence/credible intervals ("eti").
- a series of highest density confidence/credible intervals ("hdi").

**Usage**

```r
extract_distribution(
  object,
  parName = "",
  type = c("continuous", "discrete", "eti", "hdi"),
  transform = identity,
  ...
)
```
**extract_draws**

**Arguments**

- **object**: Object specifying the distribution as samples: can be a Stanfit object, a matrix (columns represents parameters, rows samples) or a vector.
- **parName**: Name of the parameter to extract.
- **type**: Indicates how the distribution is summarised.
- **transform**: Function to apply to the samples.
- **...**: Arguments to pass to `extract_pmf()`, `extract_pdf()` or `extract_ci()` depending on type.

**Value**

Dataframe

**Alternative**

This function can notably be used to prepare the data for plotting fan charts when type = "eti" or "hdi". In that case, the ggdist package offers an alternative with ggdist::stat_lineribbon().

**See Also**

- `extract_draws()` for extracting draws of an object.

**Examples**

```r
extract_distribution(runif(1e2), type = "continuous", support = c(0, 1))
```

---

### Description

Extract parameters' draws

### Usage

```r
extract_draws(obj, draws)
```

### Arguments

- **obj**: Array/Vector/Matrix of draws (cf. first dimension) or list of it.
- **draws**: Vector of draws to extract.

### Value

Dataframe with columns: Draw, Index, Value and Parameter.
Examples

```r
x <- rnorm(1e3)
X <- matrix(x, ncol = 10)
a <- array(rnorm(80), dim = c(10, 2, 2, 2))
extract_draws(x, sample(1:length(x), 10))
extract_draws(X, sample(1:nrow(X), 10))
extract_draws(a, sample(1:10, 5))
extract_draws(list(x = x, X = X, a = a), 1:10)
```

---

`extract_index_nd`  Extract multiple indices inside bracket(s) as a list

Description

Extract multiple indices inside bracket(s) as a list

Usage

```r
extract_index_nd(x, dim_names = NULL)
```

Arguments

- `x` Character vector.
- `dim_names` Optional character vector of dimension names. If `dim_names` is not NULL, if the elements of `x` don’t have the same number of indices, the missing indices will be set to NA.

Value

Dataframe with columns:

- Variable, containing `x` where brackets have been removed
- Index, a list containing values within the brackets. If `dim_names` is not NULL, Index is replaced by columns with names `dim_names` containing numeric values.

Examples

```r
extract_index_nd(c("sigma", "sigma[1]", "sigma[1, 1]", "sigma[1][2]"))
```
Extract parameters from a single draw

**Description**

Extract parameters from a single draw

**Usage**

```r
extract_parameters_from_draw(fit, param, draw)
```

**Arguments**

- `fit`: Stanfit object.
- `param`: Vector of parameter names.
- `draw`: Index of the draw to extract the parameters from.

**Value**

Dataframe

**Note**

Useful for to generate fake data.

**Alternative**

The `tidybayes` package offers an alternative to this function, for example:

```r
fit %>% tidy_draws() %>% gather_variables() %>% filter(.draw == draw & .variable %in% param)
```

However, the `tidybayes` version is less efficient as all draws and parameters are extracted and then filtered (also the draw IDs are not the same). Using `tidybayes` would be more recommended when we only want to extract specific parameters, and that it does not matter which draw are extracted (in that case using `tidybayes::spread_draws()`).
extract_pdf

*Extract probability density function from vector of samples*

**Description**

Extract probability density function from vector of samples

**Usage**

```
extract_pdf(x, support = NULL, n_density = 2^7)
```

**Arguments**

- `x` Vector of samples from a distribution.
- `support` Vector of length 2 corresponding to the range of the distribution. Can be NULL.
- `n_density` Number of equally spaced points at which the density is to be estimated (better to use a power of 2).

**Value**

Dataframe with columns: Value, Density.

**Examples**

```
extract_pdf(rnorm(1e4))
```

extract_pmf

*Extract probability mass function from vector of samples*

**Description**

Extract probability mass function from vector of samples

**Usage**

```
extract_pmf(x, support = NULL)
```

**Arguments**

- `x` Vector of samples from a distribution.
- `support` Vector of all possible values that the distribution can take. Can be NULL.

**Value**

Dataframe with columns: Value, Probability.
**factor_to_numeric**

Change the type of the column of a dataframe from factor to numeric

**Usage**

```r
factor_to_numeric(df, factor_name)
```

**Arguments**

- `df`  
  Dataframe.
- `factor_name`  
  Vector of names of factors to change to numeric.

**Value**

Same dataframe with type of the given columns changed to numeric.

**Examples**

```r
df <- data.frame(A = rep(1:5, each = 10))
df$A <- factor(df$A)
df <- factor_to_numeric(df, "A")
```

**illustrate_forward_chaining**

Illustration forward chaining

**Usage**

```r
illustrate_forward_chaining(horizon = 7, n_it = 5)
```

**Arguments**

- `horizon`  
  Prediction horizon.
- `n_it`  
  Number of iterations to display.
**Value**

Ggplot

**Examples**

`illustrate_forward_chaining()`

---

**illustrate_RPS  Illustration of the Ranked Probability Score**

**Description**

Illustration of the RPS in the case of forecasts for a discrete “Severity” score, ranging from 0 to 10. The forecast follow a (truncated between 0 and 10) Gaussian distribution, which is discretised to the nearest integer for RPS calculation.

**Usage**

`illustrate_RPS(mu = 5, sigma = 1, observed = 6)`

**Arguments**

- **mu**  
  Mean of the Gaussian forecast distribution.
- **sigma**  
  Standard deviation of the Gaussian forecast distribution.
- **observed**  
  Observed outcome.

**Details**

The RPS is the mean square error between the cumulative outcome and cumulative forecast distribution (shaded are square). The Ranked Probability Skill Score compares the RPS to a reference RPS (RPS0), \[ \text{RPSS} = 1 - \frac{\text{RPS}}{\text{RPS0}}. \] It can be interpreted as a normalised distance to a reference forecast: \[ \text{RPSS} = 0 \] means that the forecasts are not better than the reference and \[ \text{RPSS} = 1 \] corresponds to perfect forecasts.

**Value**

Ggplot

**Examples**

`illustrate_RPS()`
is_scalar

Test whether x is of length 1

Description
Test whether x is of length 1

Usage
is_scalar(x)

Arguments
x Object to be tested.

Value
Logical

Examples
is_scalar(1) # TRUE
is_scalar("a") # TRUE
is_scalar(c(1, 2)) # FALSE

is_stanfit
Test whether an object is of class "stanfit"

Description
Test whether an object is of class "stanfit"

Usage
is_stanfit(obj)

Arguments
obj Object.

Value
Boolean
**is_wholenumber**

Test whether *x* is a whole number

**Description**

- *is_wholenumber()* uses `base::round()` to test whether *x* is a whole number, it will therefore issue an error if *x* is not of mode numeric. If used in `base::stopifnot()` for example, this won't be a problem but it may be in conditionals.
- *is_scalar_wholenumber()* comes with the additional argument `check_numeric` to check whether *x* is a numeric before checking it is a whole number.

**Usage**

```r
is_wholenumber(x, tol = .Machine$double.eps^0.5)
```

```r
is_scalar_wholenumber(x, check_numeric = TRUE, ...)
```

**Arguments**

- **x** Object to be tested
- **tol** Tolerance
- **check_numeric** Whether to check whether *x* is a numeric
- **...** Arguments to pass to `is_wholenumber()`

**Value**

Logical

**Examples**

```r
is_wholenumber(1) # TRUE
is_wholenumber(1.0) # TRUE
is_wholenumber(1.1) # FALSE
is_scalar_wholenumber(1) # TRUE
is_scalar_wholenumber(c(1, 2)) # FALSE
```

---

**logit**

Logit and Inverse logit

**Description**

Logit and Inverse logit
post_pred_pval

Usage

logit(x)

inv_logit(x)

Arguments

x Numeric vector.

Value

Numeric vector.

Examples

logit(0.5)

inv_logit(0)

post_pred_pval Posterior Predictive p-value

Description

Compute and plot posterior predictive p-value (Bayesian p-value) from samples of a distribution. The simulations and observations are first summarised into a test statistics, then the test statistic of the observations is compared to the test statistic of the empirical distribution.

Usage

post_pred_pval(
    yrep,
    y,
    test_statistic = mean,
    alternative = c("two.sided", "less", "greater"),
    plot = FALSE
)

Arguments

yrep Matrix of posterior replications with rows corresponding to samples and columns to simulated observations.

y Vector of observations.

test_statistic Function of the test statistic to compute the p-value for

alternative Indicates the alternative hypothesis: must be one of "two.sided", "greater" or "less".

plot Whether to output a plot visualising the distribution of the test statistic
Value

List containing the p-value and (optionally) a ggplot

Examples

```
post_pred_pval(matrix(rnorm(1e3), ncol = 10), rnorm(10))
```

---

**PPC_group_distribution**

*Posterior Predictive Check for Stan model*

**Description**

Plot the distribution density of parameters within a same group from a single/multiple draw of the posterior distribution. In the case of a hierarchical model, we might look at the distribution of patient parameter and compare it to the prior for the population distribution.

**Usage**

```
PPC_group_distribution(obj, parName = "", nDraws = 1)
```

**Arguments**

- `obj` Matrix (rows: samples, cols: parameter) or Stanfit object.
- `parName` Name of the observation-dependent (e.g. patient-dependent) parameter to consider (optional when `obj` is a matrix).
- `nDraws` Number of draws to plot

**Value**

Ggplot of the distribution

**References**


**Examples**

```
X <- matrix(rnorm(1e3), ncol = 10)
PPC_group_distribution(X, "", 10)
```
**Description**

- `combine_prior_posterior` subsets and binds the prior and posterior dataframes.
- `plot_prior_posterior` plots posterior CI alongside prior CI.
- `compute_prior_influence` computes diagnostics of how the posterior is influenced by the prior.
- `plot_prior_influence` plots diagnostics from `compute_prior_influence`.

**Usage**

```r
combine_prior_posterior(prior, post, pars = NULL, match_exact = TRUE)

plot_prior_posterior(
  prior,
  post,
  pars = NULL,
  match_exact = TRUE,
  lb = "5%",
  ub = "95%"
)

compute_prior_influence(
  prior,
  post,
  pars = NULL,
  match_exact = TRUE,
  remove_index_prior = TRUE
)

plot_prior_influence(prior, post, pars = NULL, match_exact = TRUE)

check_model_sensitivity(prior, post, pars = NULL)
```

**Arguments**

- `prior` Dataframe of prior parameter estimates. The dataframe is expected to have columns `Variable`, `Mean`. For `plot_prior_posterior()`, the columns `5%` and `95%` should also be present. For `compute_prior_influence()` and `plot_prior_influence()`, the columns `Index` and `sd` should also be present.
- `post` Dataframe of posterior parameter estimates, with same columns as `prior`.
- `pars` Vector of parameter names to plot. Defaults to all parameters presents in `post` and `prior`.

prior_posterior

match_exact: Logical indicating whether parameters should be matched exactly (e.g. \( p \) does not match \( p[1] \)).

lb: Name of the column in prior and post corresponding to lower bound of error bar.

ub: Name of the column in prior and post corresponding to upper bound of error bar.

remove_index_prior: Whether to remove the index variable for prior except the first one. This is useful if a parameter with multiple index have the same prior distribution (e.g. with subject parameters, when prior does not contain as many subjects as post for computational reasons).

Details

- Posterior shrinkage (PostShrinkage = 1 - Var(Post) / Var(Prior)), capturing how much the model is learning. Shrinkage near 0 indicates that the data provides little information beyond the prior. Shrinkage near 1 indicates that the data is much more informative than the prior.

- 'Mahalanobis' distance between the mean posterior and the prior (DistPrior), capturing whether the prior "includes" the posterior.

Value

- combine_prior_posterior returns a dataframe with the same columns as in prior and post and a column Distribution.

- compute_prior_influence returns a dataframe with columns: Variable, Index, PostShrinkage, DistPrior.

- plot_prior_posterior and plot_prior_influence returns a ggplot object

Note

For plot_prior_posterior, parameters with the same name but different indices are plotted together. If their prior distribution is the same, it can be useful to only keep one index in prior. If not, we can use match_exact = FALSE to plot parameter[1] and parameter[2] separately.

References

### process_replications

**Extract posterior predictive distribution**

#### Description

Extract posterior predictive distribution

#### Usage

```r
process_replications(
  fit,
  idx = NULL,
  parName,
  bounds = NULL,
  type = c("continuous", "discrete", "eti", "hdi"),
  ...
)
```

#### Arguments

- **fit**
  - Stanfit object.
- **idx**
  - Dataframe for translating the indices of the parameters into more informative variable (can be NULL).
- **parName**
  - Name of the parameter to extract.
- **bounds**
  - NULL or vector of length 2 representing the bounds of the distribution if it needs to be truncated.
- **type**
  - Indicates how the distribution is summarised.
- **...**
  - Parameters to be passed to `extract_distribution()`.

#### Value

Dataframe.

### summary_statistics

**Extract summary statistics**

#### Description

Extract summary statistics

#### Usage

```r
summary_statistics(fit, pars, probs = c(0.05, 0.25, 0.5, 0.75, 0.95))
```
Arguments

- **fit**: Stanfit object.
- **pars**: Character vector of parameters to extract. Defaults to all parameters.
- **probs**: Numeric vector of quantiles to extract.

Value

Dataframe of posterior summary statistics

Alternative

The `tidybayes` package offers an alternative to this function, for example: `fit %>% tidy_draws() %>% gather_variables() %>% mean_qi()`. However, this does not provide information about Rhat or Neff, nor does it process the indexes. The `tidybayes` package is more useful for summarising the distribution of a handful of parameters (using `tidybayes::spread_draws()`).
Index

* datasets
  cbbPalette, 3
  %~% (approx_equal), 2
  approx_equal, 2
  base::round(), 16
  base::stopifnot(), 16
  cbbPalette, 3
  change_colnames, 3
  check_model_sensitivity
    (prior_posterior), 19
  combine_prior_posterior
    (prior_posterior), 19
  compute_calibration, 4
  compute_coverage (coverage), 6
  compute_prior_influence
    (prior_posterior), 19
  compute_resolution, 5
  compute_RPS, 5
  coverage, 6
  empirical_pval, 7
  extract_ci, 8
  extract_ci(), 9
  extract_distribution, 8
  extract_distribution(), 21
  extract_draws, 9
  extract_draws(), 9
  extract_index Nd, 10
  extract_parameters_from_draw, 11
  extract_pdf, 12
  extract_pdf(), 9
  extract_pmf, 12
  extract_pmf(), 9
  factor_to_numeric, 13
  illustrate_forward_chaining, 13
  illustrate_RPS, 14
  inv_logit (logit), 16
  is_scalar, 15
  is_scalar_wholenumber (is_wholenumber), 16
  is_stanfit, 15
  is_wholenumber, 16
  logit, 16
  plot_coverage (coverage), 6
  plot_prior_influence (prior_posterior), 19
  plot_prior_posterior (prior_posterior), 19
  post_pred_pval, 17
  PPC_group_distribution, 18
  prior_posterior, 19
  process_replications, 21
  stats::loess(), 4
  summary_statistics, 21