Package ‘scoringutils’

July 21, 2021

Title Utilities for Scoring and Assessing Predictions
Version 0.1.7.2
Language en-GB

Description Combines a collection of metrics and proper scoring rules
(Tilmann Gneiting & Adrian E Raftery (2007)
<doi:10.1198/016214506000001437>) with an easy to
use wrapper that can be used to automatically evaluate predictions.
Apart from proper scoring rules functions are provided to assess bias,
sharpness and calibration (Sebastian Funk, Anton Camacho, Adam J. Kucharski,
<doi:10.1371/journal.pcbi.1006785>) of forecasts.
Several types of predictions can be evaluated:
probabilistic forecasts (generally
predictive samples generated by Markov Chain Monte Carlo procedures),
quantile forecasts or point forecasts. Observed values and predictions
can be either continuous, integer, or binary. Users can either choose
to apply these rules separately in a vector/matrix format that can
be flexibly used within other packages, or they can choose to do an
automatic evaluation of their forecasts. This is implemented with
‘data.table’ and provides a consistent and very efficient framework for
evaluating various types of predictions.

License MIT + file LICENSE
Encoding UTF-8
LazyData true
Imports data.table, forcats, ggplot2, goftest, scoringRules, stats,
methods
Suggests testthat, knitr, rmarkdown
RoxygenNote 7.1.1
URL https://github.com/epiforecasts/scoringutils
BugReports https://github.com/epiforecasts/scoringutils/issues
VignetteBuilder knitr
Depends R (>= 2.10)
NeedsCompilation no

Author Nikos Bosse [aut, cre] (https://orcid.org/0000-0002-7750-5280),
  Sam Abbott [aut] (https://orcid.org/0000-0001-8057-8037),
  Johannes Bracher [ctb] (https://orcid.org/0000-0002-3777-1410),
  Joel Hellewell [ctb] (https://orcid.org/0000-0003-2683-0849),
  Sophie Meakins [ctb],
  James Munday [ctb],
  Katharine Sherratt [ctb],
  Sebastian Funk [aut]

Maintainer Nikos Bosse <nikosbosse@gmail.com>

Repository CRAN

Date/Publication 2021-07-21 09:40:02 UTC

R topics documented:

abs_error ................................................................. 3
add_quantiles ............................................................ 4
add_rel_skill_to_eval_forecasts ..................................... 5
add_sd ................................................................. 6
ae_median_quantile ..................................................... 6
ae_median_sample ....................................................... 7
bias ................................................................. 8
binary_example_data ................................................... 9
brier_score ............................................................. 10
check_equal_length ................................................... 11
check_not_null ......................................................... 11
compare_two_models ................................................... 12
continuous_example_data ............................................ 13
correlation_plot ....................................................... 13
crps ................................................................. 14
delete_columns .......................................................... 15
dss ................................................................. 15
eval_forecasts .......................................................... 16
eval_forecasts_binary .................................................. 20
eval_forecasts_sample ................................................ 22
example_quantile_forecasts_only .................................... 24
example_truth_data_only ............................................. 25
extract_from_list ...................................................... 26
geom_mean_helper ..................................................... 26
hist_PIT ............................................................ 27
hist_PIT_quantile ...................................................... 27
integer_example_data ................................................ 28
interval_coverage ..................................................... 28
interval_score ........................................................ 29
logs ............................................................... 31
merge_pred_and_obs .................................................. 32
**abs_error**

Absolute Error

**Description**

Calculate absolute error as

\[ \text{abs(true_value} - \text{prediction}) \]

**Usage**

\[ \text{abs_error(true_values, predictions)} \]
**add_quantiles**

**Arguments**

- `true_values` A vector with the true observed values of size n
- `predictions` numeric vector with predictions, corresponding to the quantiles in a second vector, 'quantiles'.

**Value**

vector with the absolute error

**Examples**

```r
ture_values <- rnorm(30, mean = 1:30)
predicted_values <- rnorm(30, mean = 1:30)
abs_error(true_values, predicted_values)
```

---

**add_quantiles**

*Add Quantiles to Predictions When Summarising*

**Description**

Helper function used within eval_forecasts

**Usage**

```
add_quantiles(dt, varnames, quantiles, by)
```

**Arguments**

- `dt` the data.table operated on
- `varnames` names of the variables for which to calculate quantiles
- `quantiles` the desired quantiles
- `by` grouping variable in 'eval_forecasts()'

**Value**

‘data.table‘ with quantiles added
**add_rel_skill_to_eval_forecasts**

*Add relative skill to eval_forecasts()*

---

**Description**

This function will only be called within `eval_forecasts` and serves to make pairwise comparisons from within that function. It uses the 'summarise_by' argument as well as the data from `eval_forecasts`. Essentially, it wraps `pairwise_comparison` and deals with the specifics necessary to work with `eval_forecasts`.

**Usage**

```r
add_rel_skill_to_eval_forecasts(
  unsummarised_scores,  # unsummarised scores to be passed from eval_forecasts
  rel_skill_metric,     # character string with the name of the metric for which a relative skill shall be computed. If equal to 'auto' (the default), then one of interval score, crps or brier score will be used where appropriate
  baseline,             # character string with the name of a model. If a baseline is given, then a scaled relative skill with respect to the baseline will be returned. By default ('NULL'), relative skill will not be scaled with respect to a baseline model.
  by,                   # character vector of columns to group scoring by. This should be the lowest level of grouping possible, i.e. the unit of the individual observation. This is important as many functions work on individual observations. If you want a different level of aggregation, you should use summarise_by to aggregate the individual scores. Also not that the pit will be computed using summarise_by instead of by
  summarise_by,        # character vector of columns to group the summary by. By default, this is equal to 'by' and no summary takes place. But sometimes you may want to to summarise over categories different from the scoring. summarise_by is also the grouping level used to compute (and possibly plot) the probability integral transform(pit).
  verbose = TRUE        # print out additional helpful messages (default is TRUE)
)
```

**Arguments**

- **unsummarised_scores**: unsummarised scores to be passed from `eval_forecasts`
- **rel_skill_metric**: character string with the name of the metric for which a relative skill shall be computed. If equal to 'auto' (the default), then one of interval score, crps or brier score will be used where appropriate
- **baseline**: character string with the name of a model. If a baseline is given, then a scaled relative skill with respect to the baseline will be returned. By default ('NULL'), relative skill will not be scaled with respect to a baseline model.
- **by**: character vector of columns to group scoring by. This should be the lowest level of grouping possible, i.e. the unit of the individual observation. This is important as many functions work on individual observations. If you want a different level of aggregation, you should use summarise_by to aggregate the individual scores. Also not that the pit will be computed using summarise_by instead of by
- **summarise_by**: character vector of columns to group the summary by. By default, this is equal to 'by' and no summary takes place. But sometimes you may want to to summarise over categories different from the scoring. summarise_by is also the grouping level used to compute (and possibly plot) the probability integral transform(pit).
- **verbose**: print out additional helpful messages (default is TRUE)
add_sd  
Add Standard Deviation to Predictions When Summarising

Description
Helper function used within eval_forecasts

Usage
add_sd(dt, varnames, by)

Arguments
- dt: the data.table operated on
- varnames: names of the variables for which to calculate the sd
- by: grouping variable in 'eval_forecasts()

Value
'data.table' with sd added

ae_median_quantile  
Absolute Error of the Median (Quantile-based Version)

Description
Absolute error of the median calculated as

$$\text{abs}(\text{true}_{\text{value}} - \text{median}_{\text{prediction}})$$

Usage
ae_median_quantile(true_values, predictions, quantiles = NULL, verbose = TRUE)

Arguments
- true_values: A vector with the true observed values of size n
- predictions: numeric vector with predictions, corresponding to the quantiles in a second vector, ‘quantiles’.
- quantiles: numeric vector that denotes the quantile for the values in ‘predictions’. Only those predictions where ‘quantiles == 0.5’ will be kept. If ‘quantiles’ is ‘NULL’, then all ‘predictions’ and ‘true_values’ will be used (this is then the same as ‘absolute_error()’)
- verbose: logical, return a warning is something unexpected happens
**ae_median_sample**

Value

vector with the scoring values

Examples

```r
true_values <- rnorm(30, mean = 1:30)
predicted_values <- rnorm(30, mean = 1:30)
ae_median_quantile(true_values, predicted_values, quantiles = 0.5)
```

---

**ae_median_sample**  
*Absolute Error of the Median (Sample-based Version)*

Description

Absolute error of the median calculated as

\[
\text{abs}(\text{true}_\text{value} - \text{median}_\text{prediction})
\]

Usage

```r
ae_median_sample(true_values, predictions)
```

Arguments

- `true_values`  
  A vector with the true observed values of size n

- `predictions`  
  nxN matrix of predictive samples, n (number of rows) being the number of data points and N (number of columns) the number of Monte Carlo samples. Alternatively, predictions can just be a vector of size n

Value

vector with the scoring values

Examples

```r
true_values <- rnorm(30, mean = 1:30)
predicted_values <- rnorm(30, mean = 1:30)
ae_median_sample(true_values, predicted_values)
```
bias

Determines bias of forecasts

Description

Determines bias from predictive Monte-Carlo samples. The function automatically recognises, whether forecasts are continuous or integer valued and adapts the Bias function accordingly.

Usage

bias(true_values, predictions)

Arguments

true_values A vector with the true observed values of size n
predictions nxN matrix of predictive samples, n (number of rows) being the number of data points and N (number of columns) the number of Monte Carlo samples

Details

For continuous forecasts, Bias is measured as

\[ B_t(P_t, x_t) = 1 - 2 \times (P_t(x_t)) \]

where \( P_t \) is the empirical cumulative distribution function of the prediction for the true value \( x_t \). Computationally, \( P_t(x_t) \) is just calculated as the fraction of predictive samples for \( x_t \) that are smaller than \( x_t \).

For integer valued forecasts, Bias is measured as

\[ B_t(P_t, x_t) = 1 - (P_t(x_t) + P_t(x_t + 1)) \]

to adjust for the integer nature of the forecasts.

In both cases, Bias can assume values between -1 and 1 and is 0 ideally.

Value

vector of length n with the biases of the predictive samples with respect to the true values.

Author(s)

Nikos Bosse <nikosbosse@gmail.com>
References


Examples

```r
## integer valued forecasts
true_values <- rpois(30, lambda = 1:30)
predictions <- replicate(200, rpois(n = 30, lambda = 1:30))
bias(true_values, predictions)

## continuous forecasts
true_values <- rnorm(30, mean = 1:30)
predictions <- replicate(200, rnorm(30, mean = 1:30))
bias(true_values, predictions)
```

---

**binary_example_data**  
*Binary Forecast Example Data*

Description

A data set with (constructed) binary predictions relevant in the 2020 UK Covid-19 epidemic.

Usage

`binary_example_data`

Format

A data frame with 346 rows and 10 columns:

- `value_date` the date for which a prediction was made
- `value_type` the target to be predicted (short form)
- `geography` the region for which a prediction was made
- `value_desc` long form description of the prediction target
- `model` name of the model that generated the forecasts
- `creation_date` date on which the forecast was made
- `horizon` forecast horizon in days
- `prediction` probability prediction that true value would be 1
- `true_value` true observed values
Details
Predictions in the data set were constructed based on the continuous example data by looking at the number of samples below the mean prediction. The outcome was constructed as whether or not the actually observed value was below or above that mean prediction. This should not be understood as sound statistical practice, but rather as a practical way to create an example data set.

brier_score

Description
Computes the Brier Score for probabilistic forecasts of binary outcomes.

Usage
brier_score(true_values, predictions)

Arguments
true_values A vector with the true observed values of size n
predictions A vector with a predicted probability that true_value = 1.

Details
The Brier score is a proper score rule that assesses the accuracy of probabilistic binary predictions. The outcomes can be either 0 or 1, the predictions must be a probability that the true outcome will be 1.

The Brier Score is then computed as the mean squared error between the probabilistic prediction and the true outcome.

\[
BrierScore = \frac{1}{N} \sum_{t=1}^{n} (prediction_t - outcome_t)^2
\]

Value
A numeric value with the Brier Score, i.e. the mean squared error of the given probability forecasts

Examples
true_values <- sample(c(0,1), size = 30, replace = TRUE)
predictions <- runif(n = 30, min = 0, max = 1)
brier_score(true_values, predictions)
check_equal_length

Description
Check whether variables all have the same length

Usage
check_equal_length(..., one_allowed = TRUE)

Arguments
... The variables to check
one_allowed logical, allow arguments of length one that can be recycled

Value
The function returns ‘NULL’, but throws an error if variable lengths differ

check_not_null

Description
Check whether a certain variable is not ‘NULL’ and return the name of that variable and the function call where the variable is missing. This function is a helper function that should only be called within other functions

Usage
check_not_null(...)

Arguments
... The variables to check

Value
The function returns ‘NULL’, but throws an error if the variable is missing.
compare_two_models  

Compare Two Models Based on Subset of Common Forecasts

Description
This function compares two models based on the subset of forecasts for which both models have made a prediction. It gets called from pairwise_comparison_one_group, which handles the comparison of multiple models on a single set of forecasts (there are no subsets of forecasts to be distinguished). pairwise_comparison_one_group in turn gets called from from pairwise_comparison which can handle pairwise comparisons for a set of forecasts with multiple subsets, e.g. pairwise comparisons for one set of forecasts, but done separately for two different forecast targets.

Usage
```
compare_two_models(scores, name_model1, name_model2, metric, test_options, by)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scores</td>
<td>A data.frame of unsummarised scores as produced by eval_forecasts</td>
</tr>
<tr>
<td>name_model1</td>
<td>character, name of the first model</td>
</tr>
<tr>
<td>name_model2</td>
<td>character, name of the model to compare against</td>
</tr>
<tr>
<td>metric</td>
<td>A character vector of length one with the metric to do the comparison on.</td>
</tr>
<tr>
<td>test_options</td>
<td>list with options to pass down to compare_two_models. To change only one of</td>
</tr>
<tr>
<td></td>
<td>the default options, just pass a list as input with the name of the argument</td>
</tr>
<tr>
<td></td>
<td>you want to change. All elements not included in the list will be set to</td>
</tr>
<tr>
<td></td>
<td>the default (so passing an empty list would result in the default options).</td>
</tr>
<tr>
<td>by</td>
<td>character vector of columns to group scoring by. This should be the lowest</td>
</tr>
<tr>
<td></td>
<td>level of grouping possible, i.e. the unit of the individual observation.</td>
</tr>
<tr>
<td></td>
<td>This is important as many functions work on individual observations. If you</td>
</tr>
<tr>
<td></td>
<td>want a different level of aggregation, you should use summarise_by to</td>
</tr>
<tr>
<td></td>
<td>aggregate the individual scores. Also not that the pit will be computed</td>
</tr>
<tr>
<td></td>
<td>using summarise_by instead of by</td>
</tr>
</tbody>
</table>

Author(s)

Johannes Bracher, <johannes.bracher@kit.edu>

Nikos Bosse <nikosbosse@gmail.com>
**continuous_example_data**

*Continuous Forecast Example Data*

**Description**


**Usage**

`continuous_example_data`

**Format**

A data frame with 13,429 rows and 10 columns:

- `value_date` the date for which a prediction was made
- `value_type` the target to be predicted (short form)
- `geography` the region for which a prediction was made
- `value_desc` long form description of the prediction target
- `model` name of the model that generated the forecasts
- `creation_date` date on which the forecast was made
- `horizon` forecast horizon in days
- `prediction` prediction value for the corresponding sample
- `sample` id for the corresponding sample
- `true_value` true observed values

**correlation_plot**

*Plot Correlation Between Metrics*

**Description**

Plots a coloured table of scores obtained using `eval_forecasts`

**Usage**

`correlation_plot(scores, select_metrics = NULL)`

**Arguments**

- `scores` A data frame of scores as produced by `eval_forecasts`
- `select_metrics` A character vector with the metrics to show. If set to NULL (default), all metrics present in `summarised_scores` will be shown
**Value**

A ggplot2 object showing a coloured matrix of correlations between metrics

**Examples**

```r
scores <- scoringutils::eval_forecasts(scoringutils::quantile_example_data)
sCORingutils::correlation_plot(scores)
```

---

**crps**

*Ranked Probability Score*

**Description**

Wrapper around the `crps_sample` function from the scoringRules package. Can be used for continuous as well as integer valued forecasts

**Usage**

```r
crps(true_values, predictions)
```

**Arguments**

| true_values | A vector with the true observed values of size n |
| predictions | nxN matrix of predictive samples, n (number of rows) being the number of data points and N (number of columns) the number of Monte Carlo samples |

**Value**

vector with the scoring values

**References**


**Examples**

```r
true_values <- rpois(30, lambda = 1:30)
predictions <- replicate(200, rpois(n = 30, lambda = 1:30))
crps(true_values, predictions)
```
**delete_columns**

*Delete Columns From a Data.table*

**Description**

take a vector of column names and delete the columns if they are present in the data.table

**Usage**

```r
delete_columns(df, cols_to_delete)
```

**Arguments**

df: A data.table or data.frame from which columns shall be deleted

cols_to_delete: character vector with names of columns to be deleted

**Value**

A data.table

---

**dss**

*Dawid-Sebastiani Score*

**Description**

Wrapper around the `dss_sample` function from the `scoringRules` package.

**Usage**

```r
dss(true_values, predictions)
```

**Arguments**

true_values: A vector with the true observed values of size n

predictions: nxN matrix of predictive samples, n (number of rows) being the number of data points and N (number of columns) the number of Monte Carlo samples

**Value**

vector with scoring values

**References**

Examples

```r
true_values <- rpois(30, lambda = 1:30)
predictions <- replicate(200, rpois(n = 30, lambda = 1:30))
dss(true_values, predictions)
```

**Description**

The function `eval_forecasts` is an easy to use wrapper of the lower level functions in the `scoringutils` package. It can be used to score probabilistic or quantile forecasts of continuous, integer-valued or binary variables.

**Usage**

```r
eval_forecasts(
  data = NULL,
  by = NULL,
  summarise_by = by,
  metrics = NULL,
  quantiles = c(),
  sd = FALSE,
  interval_score_arguments = list(weigh = TRUE, count_median_twice = FALSE,
                      separate_results = TRUE),
  pit_plots = FALSE,
  summarised = TRUE,
  verbose = TRUE,
  forecasts = NULL,
  truth_data = NULL,
  merge_by = NULL,
  compute_relative_skill = FALSE,
  rel_skill_metric = "auto",
  baseline = NULL
)
```

**Arguments**

- `data` A data.frame or data.table with the predictions and observations. Note: it is easiest to have a look at the example files provided in the package and in the examples below. The following columns need to be present:
  - `true_value` - the true observed values
  - `prediction` - predictions or predictive samples for one true value. (You only don’t need to provide a prediction column if you want to score quantile forecasts in a wide range format.)

For integer and continuous forecasts a sample column is needed:
• sample - an index to identify the predictive samples in the prediction column generated by one model for one true value. Only necessary for continuous and integer forecasts, not for binary predictions.

For quantile forecasts the data can be provided in a variety of formats. You can either use a range-based format or a quantile-based format. (You can convert between formats using `quantile_to_range_long`, `range_long_to_quantile`, `sample_to_range_long`, `sample_to_quantile`) For a quantile-format forecast you should provide:

• prediction - prediction to the corresponding quantile
• quantile - quantile to which the prediction corresponds

For a range format (long) forecast you need

• prediction - the quantile forecasts
• boundary values should be either "lower" or "upper", depending on whether the prediction is for the lower or upper bound of a given range
• range - the range for which a forecast was made. For a 50% interval the range should be 50. The forecast for the 25% quantile should have the value in the prediction column, the value of range should be 50 and the value of boundary should be "lower". If you want to score the median (i.e. range = 0), you still need to include a lower and an upper estimate, so the median has to appear twice.

Alternatively you can also provide the format in a wide range format. This format needs

• pairs of columns called something like 'upper_90' and 'lower_90', or 'upper_50' and 'lower_50', where the number denotes the interval range. For the median, you need to provide columns called 'upper_0' and 'lower_0'

`by` character vector of columns to group scoring by. This should be the lowest level of grouping possible, i.e. the unit of the individual observation. This is important as many functions work on individual observations. If you want a different level of aggregation, you should use `summarise_by` to aggregate the individual scores. Also note that the pit will be computed using `summarise_by` instead of by

`summarise_by` character vector of columns to group the summary by. By default, this is equal to 'by' and no summary takes place. But sometimes you may want to to summarise over categories different from the scoring. `summarise_by` is also the grouping level used to compute (and possibly plot) the probability integral transform (pit).

`metrics` the metrics you want to have in the output. If `NULL` (the default), all available metrics will be computed.

`quantiles` numeric vector of quantiles to be returned when summarising. Instead of just returning a mean, quantiles will be returned for the groups specified through 'summarise_by'. By default, no quantiles are returned.

`sd` if TRUE (the default is FALSE) the standard deviation of all metrics will be returned when summarising.

`interval_score_arguments` list with arguments for the calculation of the interval score. These arguments get passed down to `interval_score`, except for the argument 'count_median_twice'
that controls how the interval scores for different intervals are summed up. This should be a logical (default is FALSE) that indicates whether or not to count the median twice when summarising. This would conceptually treat the median as a 0% prediction interval, where the median is the lower as well as the upper bound. The alternative is to treat the median as a single quantile forecast instead of an interval. The interval score would then be better understood as an average of quantile scores.)

pit_plots if TRUE (not the default), pit plots will be returned. For details see pit.
summarised Summarise arguments (i.e. take the mean per group specified in group_by. Default is TRUE.
verbose print out additional helpful messages (default is TRUE)
forecasts data.frame with forecasts, that should follow the same general guidelines as the 'data' input. Argument can be used to supply forecasts and truth data independently. Default is 'NULL'.
truth_data data.frame with a column called 'true_value' to be merged with 'forecasts'
merge_by character vector with column names that 'forecasts' and 'truth_data' should be merged on. Default is 'NULL' and merge will be attempted automatically.
compute_relative_skill logical, whether or not to compute relative performance between models. If 'TRUE' (default is FALSE), then a column called 'model' must be present in the input data. For more information on the computation of relative skill, see pairwise_comparison. Relative skill will be calculated for the aggregation level specified in 'summarise_by'.
rel_skill_metric character string with the name of the metric for which a relative skill shall be computed. If equal to 'auto' (the default), then one of interval score, crps or brier score will be used where appropriate
baseline character string with the name of a model. If a baseline is given, then a scaled relative skill with respect to the baseline will be returned. By default ('NULL'), relative skill will not be scaled with respect to a baseline model.

Details

the following metrics are used where appropriate:

- Interval Score for quantile forecasts. Smaller is better. See interval_score for more information. By default, the weighted interval score is used.
- Brier Score for a probability forecast of a binary outcome. Smaller is better. See brier_score for more information.
- aem Absolute error of the median prediction
- Bias 0 is good, 1 and -1 are bad. See bias for more information.
- Sharpness Smaller is better. See sharpness for more information.
- Calibration represented through the p-value of the Anderson-Darling test for the uniformity of the Probability Integral Transformation (PIT). For integer valued forecasts, this p-value also has a standard deviation. Larger is better. See pit for more information.
• DSS Dawid-Sebastiani-Score. Smaller is better. See `dss` for more information.
• CRPS Continuous Ranked Probability Score. Smaller is better. See `crps` for more information.
• Log Score Smaller is better. Only for continuous forecasts. See `logs` for more information.

Value

A data.table with appropriate scores. For binary predictions, the Brier Score will be returned, for quantile predictions the interval score, as well as adapted metrics for calibration, sharpness and bias. For integer forecasts, Sharpness, Bias, DSS, CRPS, LogS, and pit_p_val (as an indicator of calibration) are returned. For integer forecasts, pit_sd is returned (to account for the randomised PIT), but no Log Score is returned (the internal estimation relies on a kernel density estimate which is difficult for integer-valued forecasts). If `summarise_by` is specified differently from `by`, the average score per summary unit is returned. If specified, quantiles and standard deviation of scores can also be returned when summarising.

Author(s)

Nikos Bosse <nikosbosse@gmail.com>

References


Examples

```r
## Probability Forecast for Binary Target
binary_example <- data.table::setDT(scoringutils::binary_example_data)
eval <- scoringutils::eval_forecasts(binary_example,
    summarise_by = c("model"),
    quantiles = c(0.5), sd = TRUE,
    verbose = FALSE)

## Quantile Forecasts
# wide format example (this examples shows usage of both wide formats)
range_example_wide <- data.table::setDT(scoringutils::range_example_data_wide)
range_example <- scoringutils::range_wide_to_long(range_example_wide)
# equivalent:
wide2 <- data.table::setDT(scoringutils::range_example_data_semi_wide)
range_example <- scoringutils::range_wide_to_long(wide2)
eval <- scoringutils::eval_forecasts(range_example,
    summarise_by = "model",
    quantiles = c(0.05, 0.95),
    sd = TRUE)
eval <- scoringutils::eval_forecasts(range_example)

#long format

eval <- scoringutils::eval_forecasts(scoringutils::range_example_data_long,
    summarise_by = "model",
    quantiles = c(0.05, 0.95),
    sd = TRUE)
eval <- scoringutils::eval_forecasts(scoringutils::range_example_data_long,
    summarise_by = "model",
    quantiles = c(0.05, 0.95),
    sd = TRUE)
```
## Integer Forecasts

```r
ingredient_example <- data.table::setDT(scoringutils::ingredient_example_data)
eval <- scoringutils::eval_forecasts(ingredient_example,
    summarise_by = c("model"),
    quantiles = c(0.1, 0.9),
    sd = TRUE,
    pit_plots = TRUE)
eval <- scoringutils::eval_forecasts(ingredient_example)
```

## Continuous Forecasts

```r
continuous_example <- data.table::setDT(scoringutils::continuous_example_data)
eval <- scoringutils::eval_forecasts(continuous_example)
eval <- scoringutils::eval_forecasts(continuous_example,
    quantiles = c(0.5, 0.9),
    sd = TRUE,
    summarise_by = c("model"))
```

---

**eval_forecasts_binary**  
*Evaluate forecasts in a Binary Format*

### Description

Evaluate forecasts in a Binary Format

### Usage

```r
eval_forecasts_binary(
    data,
    by,
    summarise_by,
    metrics,
    quantiles,
    sd,
    summarised,
    verbose
)
```

### Arguments

- **data**  
  A data.frame or data.table with the predictions and observations.  
  Note: it is easiest to have a look at the example files provided in the package and in the examples below.  
  The following columns need to be present:
  - `true_value` - the true observed values
  - `prediction` - predictions or predictive samples for one true value.  
    (You only don’t need to provide a prediction column if you want to score quantile forecasts in a wide range format.)
For integer and continuous forecasts a `sample` column is needed:

- `sample` - an index to identify the predictive samples in the prediction column generated by one model for one true value. Only necessary for continuous and integer forecasts, not for binary predictions.

For quantile forecasts the data can be provided in various formats. You can either use a range-based format or a quantile-based format. (You can convert between formats using `quantile_to_range_long`, `range_long_to_quantile`, `sample_to_range_long`, `sample_to_quantile`). For a quantile-format forecast you should provide:

- `prediction` - prediction to the corresponding quantile
- `quantile` - quantile to which the prediction corresponds

For a range format (long) forecast you need

- `prediction` - the quantile forecasts
- `boundary` values should be either "lower" or "upper", depending on whether the prediction is for the lower or upper bound of a given range
- `range` - the range for which a forecast was made. For a 50% interval the range should be 50. The forecast for the 25% quantile should have the value in the prediction column, the value of range should be 50 and the value of boundary should be "lower". If you want to score the median (i.e. range = 0), you still need to include a lower and an upper estimate, so the median has to appear twice.

Alternatively you can also provide the format in a wide range format. This format needs

- pairs of columns called something like 'upper_90' and 'lower_90', or 'upper_50' and 'lower_50', where the number denotes the interval range. For the median, you need to provide columns called 'upper_0' and 'lower_0'

by character vector of columns to group scoring by. This should be the lowest level of grouping possible, i.e. the unit of the individual observation. This is important as many functions work on individual observations. If you want a different level of aggregation, you should use `summarise_by` to aggregate the individual scores. Also note that the pit will be computed using `summarise_by` instead of `by`.

summarise_by character vector of columns to group the summary by. By default, this is equal to `by` and no summary takes place. But sometimes you may want to to summarise over categories different from the scoring. `summarise_by` is also the grouping level used to compute (and possibly plot) the probability integral transform (pit).

metrics the metrics you want to have in the output. If ‘NULL’ (the default), all available metrics will be computed.

quantiles numeric vector of quantiles to be returned when summarising. Instead of just returning a mean, quantiles will be returned for the groups specified through `summarise_by`. By default, no quantiles are returned.

sd if TRUE (the default is FALSE) the standard deviation of all metrics will be returned when summarising.

summarised Summarise arguments (i.e. take the mean per group specified in group_by. Default is TRUE.

verbose print out additional helpful messages (default is TRUE)
Value

A data.frame with appropriate scores. For more information see `eval_forecasts`.

Author(s)

Nikos Bosse <nikosbosse@gmail.com>

Examples

```r
# Probability Forecast for Binary Target
binary_example <- data.table::setDT(scoringutils::binary_example_data)
eval <- scoringutils::eval_forecasts(data = binary_example,
                                        summarise_by = c("model"),
                                        quantiles = c(0.5), sd = TRUE,
                                        verbose = FALSE)
```

---

`eval_forecasts_sample`  Evaluate forecasts in a Sample-Based Format (Integer or Continuous)

Description

Evaluate forecasts in a Sample-Based Format (Integer or Continuous)

Usage

```r
eval_forecasts_sample(
  data,
  by,
  summarise_by,
  metrics,
  prediction_type,
  quantiles,
  sd,
  pit_plots,
  summarised,
  verbose
)
```

Arguments

data  A data.frame or data.table with the predictions and observations. Note: it is easiest to have a look at the example files provided in the package and in the examples below. The following columns need to be present:
  - true_value - the true observed values
  - prediction - predictions or predictive samples for one true value. (You only don’t need to provide a prediction column if you want to score quantile forecasts in a wide range format.)
For integer and continuous forecasts a `sample` column is needed:

- `sample` - an index to identify the predictive samples in the prediction column generated by one model for one true value. Only necessary for continuous and integer forecasts, not for binary predictions.

For quantile forecasts the data can be provided in variety of formats. You can either use a range-based format or a quantile-based format. (You can convert between formats using `quantile_to_range_long`, `range_long_to_quantile`, `sample_to_range_long`, `sample_to_quantile`) For a quantile-format forecast you should provide:

- `prediction` - prediction to the corresponding quantile
- `quantile` - quantile to which the prediction corresponds

For a range format (long) forecast you need

- `prediction` the quantile forecasts
- `boundary` values should be either "lower" or "upper", depending on whether the prediction is for the lower or upper bound of a given range
- `range` the range for which a forecast was made. For a 50% interval the range should be 50. The forecast for the 25% quantile should have the value in the prediction column, the value of range should be 50 and the value of boundary should be "lower". If you want to score the median (i.e. `range = 0`), you still need to include a lower and an upper estimate, so the median has to appear twice.

Alternatively you can also provide the format in a wide range format. This format needs

- pairs of columns called something like 'upper_90' and 'lower_90', or 'upper_50' and 'lower_50', where the number denotes the interval range. For the median, you need to provide columns called 'upper_0' and 'lower_0'

by character vector of columns to group scoring by. This should be the lowest level of grouping possible, i.e. the unit of the individual observation. This is important as many functions work on individual observations. If you want a different level of aggregation, you should use `summarise_by` to aggregate the individual scores. Also note that the pit will be computed using `summarise_by` instead of by `by`

`summarise_by` character vector of columns to group the summary by. By default, this is equal to 'by' and no summary takes place. But sometimes you may want to to summarise over categories different from the scoring. `summarise_by` is also the grouping level used to compute (and possibly plot) the probability integral transform(pit).

`metrics` the metrics you want to have in the output. If 'NULL' (the default), all available metrics will be computed.

`prediction_type` character, should be either "continuous" or "integer"

`quantiles` numeric vector of quantiles to be returned when summarising. Instead of just returning a mean, quantiles will be returned for the groups specified through `summarise_by`. By default, no quantiles are returned.

`sd` if TRUE (the default is FALSE) the standard deviation of all metrics will be returned when summarising.
pit_plots if TRUE (not the default), pit plots will be returned. For details see pit.
summarised Summarise arguments (i.e. take the mean per group specified in group_by. Default is TRUE.
verbose print out additional helpful messages (default is TRUE)

Value
A data.table with appropriate scores. For more information see eval_forecasts

Author(s)
Nikos Bosse <nikosbosse@gmail.com>

References

Examples

```r
## Integer Forecasts
integer_example <- data.table::setDT(scoringutils::integer_example_data)
eval <- scoringutils::eval_forecasts(integer_example,
                                     summarise_by = c("model"),
                                     quantiles = c(0.1, 0.9),
                                     sd = TRUE,
                                     pit_plots = TRUE)
eval <- scoringutils::eval_forecasts(continuous_example)
```

```r
## Continuous Forecasts
continuous_example <- data.table::setDT(scoringutils::continuous_example_data)
eval <- scoringutils::eval_forecasts(continuous_example)
```

Description
A data set with predictions for different quantities relevant in the 2020 UK Covid-19 epidemic, but no true_values
Usage

example_quantile_forecasts_only

Format

A data frame with 7,581 rows and 9 columns:

- **value_date**: the date for which a prediction was made
- **value_type**: the target to be predicted (short form)
- **geography**: the region for which a prediction was made
- **model**: name of the model that generated the forecasts
- **creation_date**: date on which the forecast was made
- **quantile**: quantile of the corresponding prediction
- **prediction**: quantile predictions
- **value_desc**: long form description of the prediction target
- **horizon**: forecast horizon in days

---

**example_truth_data_only**

*Truth data only*

---

Description

A data set with truth data for different quantities relevant in the 2020 UK Covid-19 epidemic, but no predictions

Usage

example_truth_data_only

Format

A data frame with 140 rows and 5 columns:

- **value_date**: the date for which a prediction was made
- **value_type**: the target to be predicted (short form)
- **geography**: the region for which a prediction was made
- **value_desc**: long form description of the prediction target
- **true_value**: true observed values
**extract_from_list**  
**Extract Elements From a List of Lists**

**Description**

Extract corresponding elements from a list of lists.

**Usage**

```r
extract_from_list(list, what)
```

**Arguments**

- `list`: the list of lists
- `what`: character with the name of the element to extract from every individual list element of `list`

**Value**

A list with the extracted element from every sublist missing.

---

**geom_mean_helper**  
**Calculate Geometric Mean**

**Description**

Calculate Geometric Mean

**Usage**

```r
geom_mean_helper(x)
```

**Arguments**

- `x`: numeric vector of values for which to calculate the geometric mean

**Value**

the geometric mean of the values in `x`
**hist_PIT**

**PIT Histogram**

**Description**

Make a simple histogram of the probability integral transformed values to visually check whether a uniform distribution seems likely.

**Usage**

```r
hist_PIT(PIT_samples, num_bins = NULL, caption = NULL)
```

**Arguments**

- **PIT_samples**: A vector with the PIT values of size \( n \)
- **num_bins**: the number of bins in the PIT histogram.
- **caption**: provide a caption that gets passed to the plot. If not given, the square root of \( n \) will be used.

**Value**

vector with the scoring values

---

**hist_PIT_quantile**

**PIT Histogram Quantile**

**Description**

Make a simple histogram of the probability integral transformed values to visually check whether a uniform distribution seems likely.

**Usage**

```r
hist_PIT_quantile(PIT_samples, num_bins = NULL, caption = NULL)
```

**Arguments**

- **PIT_samples**: A vector with the PIT values of size \( n \)
- **num_bins**: the number of bins in the PIT histogram.
- **caption**: provide a caption that gets passed to the plot. If not given, the square root of \( n \) will be used.

**Value**

vector with the scoring values
**integer_example_data**  
*Integer Forecast Example Data*

**Description**


**Usage**

integer_example_data

**Format**

A data frame with 13,429 rows and 10 columns:

- **value_date**: the date for which a prediction was made
- **value_type**: the target to be predicted (short form)
- **geography**: the region for which a prediction was made
- **value_desc**: long form description of the prediction target
- **model**: name of the model that generated the forecasts
- **creation_date**: date on which the forecast was made
- **horizon**: forecast horizon in days
- **prediction**: prediction value for the corresponding sample
- **sample**: id for the corresponding sample
- **true_value**: true observed values

**interval_coverage**  
*Plot Interval Coverage*

**Description**

Plot interval coverage

**Usage**

interval_coverage(
    summarised_scores,
    colour = "model",
    facet_formula = NULL,
    facet_wrap_or_grid = "facet_wrap",
    scales = "free_y"
)
interval_score

Arguments

summarised_scores
Summarised scores as produced by `eval_forecasts`. Make sure that "range" is included in `summarise_by` when producing the summarised scores.

colour
According to which variable shall the graphs be coloured? Default is "model".

facet_formula
formula for facetting in ggplot. If this is `NULL` (the default), no facetting will take place.

facet_wrap_or_grid
Use ggplot2's `facet_wrap` or `facet_grid`? Anything other than "facet_wrap" will be interpreted as `facet_grid`. This only takes effect if `facet_formula` is not `NULL`.

scales
scales argument that gets passed down to ggplot. Only necessary if you make use of facetting. Default is "free_y".

Value

ggplot object with a plot of interval coverage.

Examples

```r
example1 <- scoringutils::range_example_data_long
scores <- scoringutils::eval_forecasts(example1, summarise_by = c("model", "range"))
interval_coverage(scores)
```

---

<table>
<thead>
<tr>
<th>interval_score</th>
<th>Interval Score</th>
</tr>
</thead>
</table>

Description

Proper Scoring Rule to score quantile predictions, following Gneiting and Raftery (2007). Smaller values are better.

The score is computed as

\[
score = (upper - lower) + 2/alpha \times (lower - true_value) \times 1(true_value < lower) + 2/alpha \times (true_value - upper) \times 1(true_value > upper)
\]

where $1()$ is the indicator function and alpha is the decimal value that indicates how much is outside the prediction interval. To improve usability, the user is asked to provide an interval range in percentage terms, i.e. `interval_range = 90` (percent) for a 90 percent prediction interval. Correspondingly, the user would have to provide the 5% and 95% quantiles (the corresponding alpha would then be 0.1). No specific distribution is assumed, but the range has to be symmetric (i.e you can’t use the 0.1 quantile as the lower bound and the 0.7 quantile as the upper).

The interval score is a proper scoring rule that scores a quantile forecast.
Usage

```
interval_score(
  true_values,
  lower,
  upper,
  interval_range,
  weigh = TRUE,
  separate_results = FALSE
)
```

Arguments

- **true_values**: A vector with the true observed values of size n
- **lower**: vector of size n with the lower quantile of the given range
- **upper**: vector of size n with the upper quantile of the given range
- **interval_range**: the range of the prediction intervals. i.e. if you’re forecasting the 0.05 and 0.95 quantile, the interval_range would be 90. Can be either a single number or a vector of size n, if the range changes for different forecasts to be scored. This corresponds to (100-alpha)/100 in Gneiting and Raftery (2007). Internally, the range will be transformed to alpha.
- **weigh**: if TRUE, weigh the score by alpha / 4, so it can be averaged into an interval score that, in the limit, corresponds to CRPS. Default: FALSE.
- **separate_results**: if TRUE (default is FALSE), then the separate parts of the interval score (sharpness, penalties for over- and under-prediction get returned as separate elements of a list). If you want a ‘data.frame’ instead, simply call ‘as.data.frame()’ on the output.

Value

vector with the scoring values, or a list with separate entries if separate_results is TRUE.

References


Examples

```
true_values <- rnorm(30, mean = 1:30)
interval_range = rep(90, 30)
alpha = (100 - interval_range) / 100
lower = qnorm(alpha/2, rnorm(30, mean = 1:30))
```
upper = qnorm(1 - alpha/2), rnorm(30, mean = 1:30))

interval_score(true_values = true_values, 
    lower = lower, 
    upper = upper, 
    interval_range = interval_range)

interval_score(true_values = c(true_values, NA), 
    lower = c(lower, NA), 
    upper = c(NA, upper), 
    separate_results = TRUE, 
    interval_range = 90)

---

**Description**

Wrapper around the `logs_sample` function from the `scoringRules` package. Used to score continuous predictions. While the Log Score is in theory also applicable to integer forecasts, the problem lies in the implementation: The Log Score needs a kernel density estimation, which is not well defined with integer-valued Monte Carlo Samples. The Log Score can be used for specific integer valued probability distributions. See the `scoringRules` package for more details.

**Usage**

`logs(true_values, predictions)`

**Arguments**

`true_values` A vector with the true observed values of size n

`predictions` nxN matrix of predictive samples, n (number of rows) being the number of data points and N (number of columns) the number of Monte Carlo samples

**Value**

vector with the scoring values

**References**


**Examples**

`true_values <- rpois(30, lambda = 1:30)`

`predictions <- replicate(200, rpois(n = 30, lambda = 1:30))`

`logs(true_values, predictions)`
merge_pred_and_obs  Merge Forecast Data And Observations

Description
The function more or less provides a wrapper around `merge` that aims to handle the merging well if additional columns are present in one or both data sets. If in doubt, you should probably merge the data sets manually.

Usage
```r
merge_pred_and_obs(
  forecasts,
  observations,
  join = c("left", "full", "right"),
  by = NULL
)
```

Arguments
- **forecasts**: data.frame with the forecast data (as can be passed to `eval_forecasts`).
- **observations**: data.frame with the observations.
- **join**: character, one of `c("left", "full", "right")`. Determines the type of the join. Usually, a left join is appropriate, but sometimes you may want to do a full join to keep dates for which there is a forecast, but no ground truth data.
- **by**: character vector that denotes the columns by which to merge. Any value that is not a column in observations will be removed.

Value
a data.frame with forecasts and observations

mse  Mean Squared Error

Description
Mean Squared Error MSE of point forecasts. Calculated as

\[
\text{mse} = \text{mean}( (\text{true values} - \text{predicted values})^2 )
\]

Usage
```r
mse(true_values, predictions)
```
pairwise_comparison

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true_values</td>
<td>A vector with the true observed values of size n</td>
</tr>
<tr>
<td>predictions</td>
<td>A vector with predicted values of size n</td>
</tr>
</tbody>
</table>

Value

vector with the scoring values

Examples

```r
true_values <- rnorm(30, mean = 1:30)
predicted_values <- rnorm(30, mean = 1:30)
mse(true_values, predicted_values)
```

Description

Make pairwise comparisons between models. The code for the pairwise comparisons is inspired by an implementation by Johannes Bracher.

The implementation of the permutation test follows the function permutationTest from the ‘surveillance’ package by Michael Höhle, Andrea Riebler and Michaela Paul.

Usage

```r
pairwise_comparison(
  scores,                      
  metric = "interval_score",  
  test_options = list(oneSided = FALSE, test_type = c("non_parametric", "permutation"),
                       n_permutations = 999),
  baseline = NULL,
  by = NULL,
  summarise_by = c("model")
)
```

Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scores</td>
<td>A data.frame of unsummarised scores as produced by eval_forecasts</td>
</tr>
<tr>
<td>metric</td>
<td>A character vector of length one with the metric to do the comparison on.</td>
</tr>
<tr>
<td>test_options</td>
<td>list with options to pass down to compare_two_models. To change only one of the default options, just pass a list as input with the name of the argument you want to change. All elements not included in the list will be set to the default (so passing an empty list would result in the default options).</td>
</tr>
<tr>
<td>baseline</td>
<td>character vector of length one that denotes the baseline model against which to compare other models.</td>
</tr>
</tbody>
</table>
by character vector of columns to group scoring by. This should be the lowest level of grouping possible, i.e. the unit of the individual observation. This is important as many functions work on individual observations. If you want a different level of aggregation, you should use summarise_by to aggregate the individual scores. Also note that the pit will be computed using summarise_by instead of by summarise_by character vector of columns to group the summary by. By default, this is equal to 'by' and no summary takes place. But sometimes you may want to to summarise over categories different from the scoring. summarise_by is also the grouping level used to compute (and possibly plot) the probability integral transform(pit).

Value

A ggplot2 object with a coloured table of summarised scores

Author(s)

Johannes Bracher, https://jbracher.github.io/
Nikos Bosse
Nikos Bosse <nikosbosse@gmail.com>
Johannes Bracher, <johannes.bracher@kit.edu>

Examples

df <- data.frame(model = rep(c("model1", "model2", "model3"), each = 10),
  date = as.Date("2020-01-01") + rep(1:5, each = 2),
  location = c(1, 2),
  interval_score = (abs(rnorm(30)));
  aem = (abs(rnorm(30))))

res <- scoringutils::pairwise_comparison(df,
  baseline = "model1")
scoringutils::plot_pairwise_comparison(res)

eval <- scoringutils::eval_forecasts(scoringutils::range_example_data_long)
pairwise <- pairwise_comparison(eval, summarise_by = c("model"))

---

pairwise_comparison_one_group

Do Pairwise Comparison for one Set of Forecasts

Description

This function does the pairwise comparison for one set of forecasts, but multiple models involved. It gets called from pairwise_comparison. pairwise_comparison splits the data into arbitrary subgroups specified by the user (e.g. if pairwise comparison should be done separately for different forecast targets) and then the actual pairwise comparison for that subgroup is managed from pairwise_comparison_one_group. In order to actually do the comparison between two models over a subset of common forecasts it calls compare_two_models.
Usage
	pairwise_comparison_one_group(
		scores,
		metric,
		test_options,
		baseline,
		by,
		summarise_by
	)

Arguments

scores A data.frame of unsummarised scores as produced by \texttt{eval_forecasts}

metric A character vector of length one with the metric to do the comparison on.

test_options list with options to pass down to \texttt{compare_two_models}. To change only one of
the default options, just pass a list as input with the name of the argument you
want to change. All elements not included in the list will be set to the default
(so passing an empty list would result in the default options).

baseline character vector of length one that denotes the baseline model against which to
compare other models.

by character vector of columns to group scoring by. This should be the lowest
level of grouping possible, i.e. the unit of the individual observation. This is
important as many functions work on individual observations. If you want a
different level of aggregation, you should use \texttt{summarise_by} to aggregate the
individual scores. Also not that the pit will be computed using \texttt{summarise_by}
instead of \texttt{by}

summarise_by character vector of columns to group the summary by. By default, this is equal to
‘by’ and no summary takes place. But sometimes you may want to to summarise
over categories different from the scoring. \texttt{summarise_by} is also the grouping
level used to compute (and possibly plot) the probability integral transform(pit).

\begin{table}
\begin{tabular}{ll}
\textbf{pit} & \textit{Probability Integral Transformation} \\
\end{tabular}
\end{table}

Description

Uses a Probability Integral Transformation (PIT) (or a randomised PIT for integer forecasts) to
assess the calibration of predictive Monte Carlo samples. Returns a p-values resulting from an
Anderson-Darling test for uniformity of the (randomised) PIT as well as a PIT histogram if speci-
fied.
Usage

\[
pit(
    true_values,
    predictions,
    plot = TRUE,
    full_output = FALSE,
    n_replicates = 50,
    num_bins = NULL,
    verbose = FALSE
)
\]

Arguments

- **true_values**: A vector with the true observed values of size \( n \)
- **predictions**: \( nxN \) matrix of predictive samples, \( n \) (number of rows) being the number of data points and \( N \) (number of columns) the number of Monte Carlo samples
- **plot**: logical. If TRUE, a histogram of the PIT values will be returned as well
- **full_output**: return all individual \( p \)-values and computed \( u_t \) values for the randomised PIT. Usually not needed.
- **n_replicates**: the number of tests to perform, each time re-randomising the PIT
- **num_bins**: the number of bins in the PIT histogram (if \( plot == TRUE \)). If not given, the square root of \( n \) will be used
- **verbose**: if TRUE (default is FALSE) more error messages are printed. Usually, this should not be needed, but may help with debugging.

Details

Calibration or reliability of forecasts is the ability of a model to correctly identify its own uncertainty in making predictions. In a model with perfect calibration, the observed data at each time point look as if they came from the predictive probability distribution at that time.

Equivalently, one can inspect the probability integral transform of the predictive distribution at time \( t \),

\[
    u_t = F_t(x_t)
\]

where \( x_t \) is the observed data point at time \( t \), \( t = 1, \ldots, n \), \( n \) being the number of forecasts, and \( SF_t \) is the (continuous) predictive cumulative probability distribution at time \( t \). If the true probability distribution of outcomes at time \( t \) is \( G_t \) then the forecasts \( eqnF_t \) are said to be ideal if \( eqnF_t = G_t \) at all times \( t \). In that case, the probabilities \( ut \) are distributed uniformly.

In the case of discrete outcomes such as incidence counts, the PIT is no longer uniform even when forecasts are ideal. In that case a randomised PIT can be used instead:

\[
    u_t = P_t(k_t) + v \ast (P_t(k_t) - P_t(k_t - 1))
\]

where \( k_t \) is the observed count, \( P_t(x) \) is the predictive cumulative probability of observing incidence \( k \) at time \( t \), \( eqnP_t (-1) = 0 \) by definition and \( v \) is standard uniform and independent of \( k \). If \( P_t \) is the true cumulative probability distribution, then \( ut \) is standard uniform.
The function checks whether integer or continuous forecasts were provided. It then applies the (randomised) probability integral and tests the values $u_t$ for uniformity using the Anderson-Darling test.

As a rule of thumb, there is no evidence to suggest a forecasting model is miscalibrated if the p-value found was greater than a threshold of $p \geq 0.1$, some evidence that it was miscalibrated if $0.01 < p < 0.1$, and good evidence that it was miscalibrated if $p \leq 0.01$. However, the AD-p-values may be overly strict and there actual usefulness may be questionable. In this context it should be noted, though, that uniformity of the PIT is a necessary but not sufficient condition of calibration.

Value

a list with the following components:

- **p_value**: p-value of the Anderson-Darling test on the PIT values. In case of integer forecasts, this will be the mean p_value from the 'n_replicates' replicates
- **sd**: standard deviation of the p_value returned. In case of continuous forecasts, this will be NA as there is only one p_value returned.
- **hist_PIT**: a plot object with the PIT histogram. Only returned if plot == TRUE. Call plot(PIT(...)$hist_PIT) to display the histogram.
- **p_values**: all p_values generated from the Anderson-Darling tests on the (randomised) PIT. Only returned if full_output = TRUE
- **u**: the $u_t$ values internally computed. Only returned if full_output = TRUE

References


Examples

```r
## continuous predictions
true_values <- rnorm(30, mean = 1:30)
predictions <- replicate(200, rnorm(n = 30, mean = 1:30))
pit(true_values, predictions)

## integer predictions
true_values <- rpois(100, lambda = 1:100)
predictions <- replicate(5000, rpois(n = 100, lambda = 1:100))
pit(true_values, predictions, n_replicates = 5)
```
Description

Wrapper around 'pit()' for use in data.frames

Usage

pit_df(
  data,
  plot = TRUE,
  full_output = FALSE,
  n_replicates = 100,
  num_bins = NULL,
  verbose = FALSE
)

Arguments

data: a data.frame with the following columns: 'true_value', 'prediction', 'sample'
plot: logical. If TRUE, a histogram of the PIT values will be returned as well
full_output: return all individual p_values and computed u_t values for the randomised PIT. Usually not needed.
n_replicates: the number of tests to perform, each time re-randomising the PIT
num_bins: the number of bins in the PIT histogram (if plot == TRUE) If not given, the square root of n will be used
verbose: if TRUE (default is FALSE) more error messages are printed. Usually, this should not be needed, but may help with debugging.

Details

see pit

Value

a list with the following components:

- data: the input data.frame (not including rows where prediction is 'NA'), with added columns 'pit_p_val' and 'pit_sd'
- hist_PIT: a plot object with the PIT histogram. Only returned if plot == TRUE. Call plot(PIT(...)$hist_PIT) to display the histogram.
- p_values: all p_values generated from the Anderson-Darling tests on the (randomised) PIT. Only returned if full_output = TRUE
- u: the u_t values internally computed. Only returned if full_output = TRUE
References


Examples

```r
example <- scoringutils::continuous_example_data
result <- pit_df(example, full_output = TRUE)
```

---

**pit_df_fast**  
*Probability Integral Transformation (data.frame Format, fast version)*

**Description**

Wrapper around ‘pit()’ for fast use in data.frames. This version of the pit does not do allow any plotting, but can iterate over categories in a data.frame as specified in the ‘by’ argument.

**Usage**

`pit_df_fast(data, n_replicates = 100, by = by)`

**Arguments**

- **data**
  a data.frame with the following columns: ‘true_value’, ‘prediction’, ‘sample’
- **n_replicates**
  the number of tests to perform, each time re-randomising the PIT
- **by**
  character vector with categories to iterate over

**Details**

see [pit](#)

**Value**

the input data.frame (not including rows where prediction is ‘NA‘), with added columns ‘pit_p_val’ and ‘pit_sd’

**References**


**Examples**

```r
example <- scoringutils::continuous_example_data
result <- pit_df(example, full_output = TRUE)
```
plot_pairwise_comparison

Plot Heatmap of Pairwise Comparisons

Description

Creates a heatmap of the ratios or pvalues from a pairwise comparison between models

Usage

plot_pairwise_comparison(
  comparison_result,
  type = c("mean_scores_ratio", "pval", "together"),
  smaller_is_good = TRUE,
  facet_formula = NULL,
  scales = "free_y",
  ncol = NULL,
  facet_wrap_or_grid = "facet_wrap"
)

Arguments

comparison_result
  A data.frame as produced by pairwise_comparison

type
  character vector of length one that is either "mean_scores_ratio" or "pval". This
denotes whether to visualise the ratio or the p-value of the pairwise comparison.
  Default is "mean_scores_ratio"

smaller_is_good
  logical (default is TRUE) that indicates whether smaller or larger values are to
  be interpreted as 'good' (as you could just invert the mean scores ratio)

facet_formula
  facetting formula passed down to ggplot. Default is NULL

scales
  scales argument that gets passed down to ggplot. Only necessary if you make
  use of facetting. Default is "free_y"

ncol
  Number of columns for facet wrap. Only relevant if facet_formula is given
  and facet_wrap_or_grid == "facet_wrap"

facet_wrap_or_grid
  Use ggplot2's facet_wrap or facet_grid? Anything other than "facet_wrap"
  will be interpreted as facet_grid. This only takes effect if facet_formula is
  not NULL

Examples

df <- data.frame(model = rep(c("model1", "model2", "model3"), each = 10),
  id = rep(1:10),
  interval_score = abs(rnorm(30, mean = rep(c(1, 1.3, 2), each = 10))),
  aem = (abs(rnorm(30))))
data <- scoringutils::quantile_example_data
scores <- scoringutils::eval_forecasts(data)
pairwise <- pairwise_comparison(scores,
    summarise_by = "value_desc")
scoringutils::plot_pairwise_comparison(pairwise,
    facet_formula = ~ value_desc,
    scales = "fixed")

plot_predictions

Plot Predictions vs True Values

Description

Make a plot of observed and predicted values

Usage

plot_predictions(
    data = NULL,
    forecasts = NULL,
    truth_data = NULL,
    merge_by = NULL,
    x = "date",
    filter_truth = list(),
    filter_forecasts = list(),
    filter_both = list(),
    range = c(0, 50, 90),
    facet_formula = NULL,
    facet_wrap_or_grid = "facet_wrap",
    ncol = NULL,
    scales = "free_y",
    allow_truth_without_pred = FALSE,
    remove_from_truth = c("model", "forecaster", "quantile", "prediction", "sample",
    "interval"),
    xlab = x,
    ylab = "True and predicted values",
    verbose = TRUE
)

Arguments

data a data.frame that follows the same specifications outlined in eval_forecasts.
The data.frame needs to have columns called "true_value", "prediction" and then
either a column called sample, or one called "quantile" or two columns called
"range" and "boundary". Internally, these will be separated into a truth and
forecast data set in order to be able to apply different filtering to truth data and
forecasts. Alternatively you can directly provide a separate truth and forecasts
data frame as input. These data sets, however, need to be mergeable, in order to connect forecasts and truth data for plotting.

forecasts data frame with forecasts, that should follow the same general guidelines as the ‘data’ input. Argument can be used to supply forecasts and truth data independently. Default is ‘NULL’.

truth_data data frame with a column called ‘true_value’ on the x-axis. Usually, this will be “date”, but it can be anything else.

merge_by character vector with column names that ‘forecasts’ and ‘truth_data’ should be merged on. Default is ‘NULL’ and merge will be attempted automatically.

x character vector of length one that denotes the name of the variable

filter_truth a list with character strings that are used to filter the truth data. Every element is parsed as an expression and evaluated in order to filter the truth data.

filter_forecasts a list with character strings that are used to filter the forecasts data. Every element is parsed as an expression and evaluated in order to filter the forecasts data.

filter_both same as ‘filter_truth’ and ‘filter_forecasts’, but applied to both data sets for convenience.

range numeric vector indicating the interval ranges to plot. If 0 is included in range, the median prediction will be shown.

facet_formula formula for facetting in ggplot. If this is NULL (the default), no facetting will take place

facet_wrap_or_grid Use ggplot2’s facet_wrap or facet_grid? Anything other than “facet_wrap” will be interpreted as facet_grid. This only takes effect if facet_formula is not NULL

ncol Number of columns for facet wrap. Only relevant if facet_formula is given and facet_wrap_or_grid == "facet_wrap"

scales scales argument that gets passed down to ggplot. Only necessary if you make use of facetting. Default is "free_y"

allow_truth_without_pred logical, whether or not to allow instances where there is truth data, but no forecast. If ‘FALSE’ (the default), these get filtered out.

remove_from_truth character vector of columns to remove from the truth data. The reason these columns are removed is that sometimes different models or forecasters don’t cover the same periods. Removing these columns from the truth data makes sure that nevertheless all available truth data is plotted (instead of having different true values depending on the period covered by a certain model).

xlab Label for the x-axis. Default is the variable name on the x-axis

ylab Label for the y-axis. Default is "True and predicted values"

verbose print out additional helpful messages (default is TRUE)

Value ggplot object with a plot of true vs predicted values
quantile_bias

Determines Bias of Quantile Forecasts

Description

Determines bias from quantile forecasts. For an increasing number of quantiles this measure converges against the sample based bias version for integer and continuous forecasts.

Usage

quantile_bias(range, lower, upper, true_value)

Arguments

range vector of corresponding size with information about the width of the central prediction interval
lower vector of length corresponding to the number of central prediction intervals that holds predictions for the lower bounds of a prediction interval
upper vector of length corresponding to the number of central prediction intervals that holds predictions for the upper bounds of a prediction interval
true_value a single true value

Details

For quantile forecasts, bias is measured as

\[ B_t = (1 - 2 \cdot \max\{i | q_{t,i} \in Q_t \land q_{t,i} \leq x_t\})1(x_t \leq q_{t,0.5}) + (1 - 2 \cdot \min\{i | q_{t,i} \in Q_t \land q_{t,i} \geq x_t\})1(x_t \geq q_{t,0.5}). \]
where $Q_t$ is the set of quantiles that form the predictive distribution at time $t$. They represent our belief about what the true value $x_t$ will be. For consistency, we define $Q_t$ such that it always includes the element $q_{t,0} = -\infty$ and $q_{t,1} = \infty$. $1()$ is the indicator function that is 1 if the condition is satisfied and $0$ otherwise. In clearer terms, $B_t$ is defined as the maximum percentile rank for which the corresponding quantile is still below the true value, if the true value is smaller than the median of the predictive distribution. If the true value is above the median of the predictive distribution, then $B_t$ is the minimum percentile rank for which the corresponding quantile is still larger than the true value. If the true value is exactly the median, both terms cancel out and $B_t$ is zero. For a large enough number of quantiles, the percentile rank will equal the proportion of predictive samples below the observed true value, and this metric coincides with the one for continuous forecasts.

Bias can assume values between -1 and 1 and is 0 ideally.

**Value**

scalar with the quantile bias for a single quantile prediction

**Author(s)**

Nikos Bosse <nikosbosse@gmail.com>

**Examples**

```r
lower <- c(6341.000, 6329.500, 6087.014, 5703.500, 5451.000, 5340.500, 4821.996, 4709.000, 4341.500, 4006.250, 1127.000, 705.500)
upper <- c(6341.000, 6352.500, 6594.986, 6978.500, 7231.000, 7341.500, 7860.004, 7973.000, 8340.500, 8675.750, 11555.000, 11976.500)
range <- c(0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 95, 98)
true_value <- 8062
quantile_bias(lower = lower, upper = upper, range = range, true_value = true_value)
```

---

**quantile_coverage**

**Plot Quantile Coverage**

**Description**

Plot quantile coverage
quantile_example_data

Usage

quantile_coverage(
  summarised_scores,
  colour = "model",
  facet_formula = NULL,
  facet_wrap_or_grid = "facet_wrap",
  scales = "free_y"
)

Arguments

summarised_scores
  Summarised scores as produced by `eval_forecasts`. Make sure that "quantile" is included in `summarise_by` when producing the summarised scores

colour
  According to which variable shall the graphs be coloured? Default is "model".

facet_formula
  formula for facetting in `ggplot`. If this is `NULL` (the default), no facetting will take place

facet_wrap_or_grid
  Use `ggplot2`'s `facet_wrap` or `facet_grid`? Anything other than "facet_wrap" will be interpreted as `facet_grid`. This only takes effect if `facet_formula` is not `NULL`

scales
  scales argument that gets passed down to `ggplot`. Only necessary if you make use of facetting. Default is "free_y"

Value

ggplot object with a plot of interval coverage

Examples

```r
example1 <- scoringutils::range_example_data_long
scores <- scoringutils::eval_forecasts(example1, summarise_by = c("model", "quantile"))
quantile_coverage(scores)
```

quantile_example_data  Quantile Example Data

Description

A data set with predictions for different quantities relevant in the 2020 UK Covid-19 epidemic.

Usage

quantile_example_data
**Format**

A data frame with 5,152 rows and 10 columns:

- **value_date** the date for which a prediction was made
- **value_type** the target to be predicted (short form)
- **geography** the region for which a prediction was made
- **value_desc** long form description of the prediction target
- **true_value** true observed values
- **model** name of the model that generated the forecasts
- **creation_date** date on which the forecast was made
- **quantile** quantile of the corresponding prediction
- **prediction** quantile predictions
- **horizon** forecast horizon in days

---

**quantile_to_long**  
*Pivot Range Format Forecasts From Wide to Long Format*

**Description**

Legacy function that will not be supported in future updates.

**Usage**

```r
quantile_to_long(data)
```

**Arguments**

- **data** a data.frame following the specifications from `eval_forecasts` for quantile forecasts. For an example, see `range_example_data_long`

**Value**

- a data.frame in long format
quantile_to_range  

Change Data from a Plain Quantile Format to a Long Range Format

Description

Legacy function that will not be supported in future updates.

Usage

quantile_to_range(data, keep_quantile_col = FALSE)

Arguments

data a data.frame following the specifications shown in the example range_example_data_long
keep_quantile_col keep the quantile column in the final output after transformation (default is FALSE)

Value

a data.frame in long format

quantile_to_range_long

Change Data from a Plain Quantile Format to a Long Range Format

Description

Transform data from a format that uses quantiles only to one that uses interval ranges to denote quantiles.

Given a data.frame that follows the structure shown in quantile_example_data, the function outputs the same data in a long format as (as shown in range_example_data_long).

Usage

quantile_to_range_long(data, keep_quantile_col = TRUE)

Arguments

data a data.frame following the specifications shown in the example range_example_data_long
keep_quantile_col keep the quantile column in the final output after transformation (default is FALSE)
Value

a data.frame in a long interval range format

Examples

quantile <- scoringutils::quantile_example_data
long <- scoringutils::quantile_to_range_long(quantile)

quantile_to_wide  Pivot Range Format Forecasts From Long to Wide Format

Description

Legacy function that will not be supported in future updates.

Usage

quantile_to_wide(data)

Arguments

data  a data.frame following the specifications from eval_forecasts for quantile forecasts. For an example, see range_example_data_long

Value

a data.frame in wide format

range_example_data_long  Range Forecast Example Data (Long Format)

Description

A data set with predictions with different interval ranges relevant in the 2020 UK Covid-19 epidemic.

Usage

range_example_data_long
Format

A data frame with 5,419 rows and 12 columns:

- **value_date**: the date for which a prediction was made
- **value_type**: the target to be predicted (short form)
- **geography**: the region for which a prediction was made
- **value_desc**: long form description of the prediction target
- **true_value**: true observed values
- **model**: name of the model that generated the forecasts
- **creation_date**: date on which the forecast was made
- **prediction**: value for the lower or upper bound of the given prediction interval
- **horizon**: forecast horizon in days
- **boundary**: indicate lower or upper bound of prediction interval
- **range**: range of the corresponding prediction interval

Description

A data set with predictions with different interval ranges relevant in the 2020 UK Covid-19 epidemic.

Usage

range_example_data_semi_wide

Format

A data frame with 5,419 rows and 12 columns:

- **value_date**: the date for which a prediction was made
- **value_type**: the target to be predicted (short form)
- **geography**: the region for which a prediction was made
- **value_desc**: long form description of the prediction target
- **true_value**: true observed values
- **model**: name of the model that generated the forecasts
- **creation_date**: date on which the forecast was made
- **horizon**: forecast horizon in days
- **range**: range of the corresponding prediction interval
- **lower**: prediction for the lower bound of the corresponding interval
- **upper**: prediction for the upper bound of the corresponding interval
Range Forecast Example Data (Wide Format)

Description

A data set with predictions with different interval ranges relevant in the 2020 UK Covid-19 epidemic.

Usage

range_example_data_wide

Format

A data frame with 346 rows and 28 columns:

- **value_date**: the date for which a prediction was made
- **value_type**: the target to be predicted (short form)
- **geography**: the region for which a prediction was made
- **value_desc**: long form description of the prediction target
- **true_value**: true observed values
- **model**: name of the model that generated the forecasts
- **creation_date**: date on which the forecast was made
- **horizon**: forecast horizon in days
- **lower_0**: prediction for the lower bound of the 0% interval range (median)
- **lower_10**: prediction for the lower bound of the 10% interval range
- **lower_20**: prediction for the lower bound of the 20% interval range
- **lower_30**: prediction for the lower bound of the 30% interval range
- **lower_40**: prediction for the lower bound of the 40% interval range
- **lower_50**: prediction for the lower bound of the 50% interval range
- **lower_60**: prediction for the lower bound of the 60% interval range
- **lower_70**: prediction for the lower bound of the 70% interval range
- **lower_80**: prediction for the lower bound of the 80% interval range
- **lower_90**: prediction for the lower bound of the 90% interval range
- **upper_0**: prediction for the upper bound of the 0% interval range
- **upper_10**: prediction for the upper bound of the 1% interval range
- **upper_20**: prediction for the upper bound of the 20% interval range
- **upper_30**: prediction for the upper bound of the 30% interval range
- **upper_40**: prediction for the upper bound of the 40% interval range
range_long_to_quantile

Change Data from a Range Format to a Quantile Format

Description

Transform data from a format that uses interval ranges to denote quantiles to a format that uses quantiles only.

Given a data.frame that follows the structure shown in `range_example_data_long`, the function outputs the same data in a long format as (as shown in `range_example_data_long`). This can be useful e.g. for plotting. If you’re data.frame is in a different format, consider running `range_long_to_wide` first.

Usage

```r
range_long_to_quantile(data, keep_range_col = FALSE)
```

Arguments

- `data` a data.frame following the specifications from `eval_forecasts` for quantile forecasts. For an example, see `range_example_data_long`
- `keep_range_col` keep the range and boundary columns after transformation (default is FALSE)

Value

a data.frame in a plain quantile format

Examples

```r
wide <- scoringutils::range_example_data_wide
good <- scoringutils::range_wide_to_long(wide)
plain_quantile <- range_long_to_quantile(good)
```
range_long_to_wide  

Pivot Range Format Forecasts From Long to Wide Format

Description

Given a data.frame that follows the structure shown in `range_example_data_long`, the function outputs the same data in a long format as (as shown in `range_example_data_wide`). This can be useful e.g. for plotting.

Usage

```
range_long_to_wide(data)
```

Arguments

data  
a data.frame following the specifications from `eval_forecasts`) for quantile forecasts. For an example, see `range_example_data_long`)

Value

a data.frame in wide format

Examples

```
long <- scoringutils::range_example_data_long
wide <- scoringutils::range_long_to_wide(long)
```

range_plot  

Plot Metrics by Range of the Prediction Interval

Description

Visualise the metrics by range, e.g. if you are interested how different interval ranges contribute to the overall interval score, or how sharpness changes by range.

Usage

```
range_plot(
  scores,
  y = "interval_score",
  x = "model",
  colour = "range",
  facet_formula = NULL,
  scales = "free_y",
  ncol = NULL,
) ```
range_plot

```r
facet_wrap_or_grid = "facet_wrap",
  xlab = x,
  ylab = y
```

### Arguments

- **scores**: A data.frame of scores based on quantile forecasts as produced by `eval_forecasts`. Note that "range" must be included in the summarise_by argument when running `eval_forecasts`.
- **y**: The variable from the scores you want to show on the y-Axis. This could be something like "interval_score" (the default) or "sharpness".
- **x**: The variable from the scores you want to show on the x-Axis. Usually this will be "model".
- **colour**: Character vector of length one used to determine a variable for colouring dots. The Default is "range".
- **facet_formula**: faceting formula passed down to ggplot. Default is NULL.
- **scales**: scales argument that gets passed down to ggplot. Only necessary if you make use of faceting. Default is "free_y".
- **ncol**: Number of columns for facet wrap. Only relevant if `facet_formula` is given and `facet_wrap_or_grid == "facet_wrap"`.
- **facet_wrap_or_grid**: Use ggplot2's `facet_wrap` or `facet_grid`? Anything other than "facet_wrap" will be interpreted as `facet_grid`. This only takes effect if `facet_formula` is not NULL.
- **xlab**: Label for the x-axis. Default is the variable name on the x-axis.
- **ylab**: Label for the y-axis. Default is "WIS contributions".

### Value

A ggplot2 object showing a contributions from the three components of the weighted interval score.

### Examples

```r
scores <- scoringutils::eval_forecasts(scoringutils::quantile_example_data, summarise_by = c("model", "value_desc", "range"))

scores <- scoringutils::eval_forecasts(scoringutils::range_example_data_long, summarise_by = c("model", "value_desc", "range"))
scoringutils::range_plot(scores, x = "model", facet_formula = ~ value_desc)

# visualise sharpness instead of interval score
scoringutils::range_plot(scores, y = "sharpness", x = "model", facet_formula = ~value_desc)

# we saw above that sharpness values crossed. Let's look at the unweighted WIS
scores <- scoringutils::eval_forecasts(scoringutils::range_example_data_long, interval_score_arguments = list(weigh = FALSE),
```
scoringutils::range_plot(scores, y = "sharpness", x = "model",
    facet_formula = ~value_desc)

range_to_quantile  Pivot Change Data from a Range Format to a Quantile Format

Description
Legacy function that will not be supported in future updates.

Usage
range_to_quantile(data, keep_range_col = FALSE)

Arguments
data  a data.frame following the specifications from eval_forecasts for quantile forecasts. For an example, see range_example_data_long
keep_range_col  keep the range and boundary columns after transformation (default is FALSE)

Value
a data.frame in long format

range_wide_to_long  Pivot Range Format Forecasts From Wide to Long Format

Description
Given a data.frame that follows the structure shown in range_example_data_wide, the function outputs the same data in a long format as (as shown in range_example_data_long). This can be useful e.g. for plotting.

Usage
range_wide_to_long(data)

Arguments
data  a data.frame following the specifications from eval_forecasts for quantile forecasts. For an example, see range_example_data_wide

Value
a data.frame in long format
**sample_to_quantile**

**Examples**

```r
wide <- scoringutils::range_example_data_wide
ground <- scoringutils::range_wide_to_long(wide)
```

---

**Description**

Transform data from a format that is based on predictive samples to a format based on plain quantiles.

**Usage**

```r
sample_to_quantile(data, quantiles = c(0.05, 0.25, 0.5, 0.75, 0.95), type = 7)
```

**Arguments**

- `data`: a data.frame with samples
- `quantiles`: a numeric vector of quantiles to extract
- `type`: type argument passed down to the quantile function. For more information, see `quantile`

**Value**

a data.frame in a long interval range format

**Examples**

```r
e example_data <- scoringutils::integer_example_data
quantile_data <- scoringutils::sample_to_quantile(example_data)
```
sample_to_range_long

Change Data from a Sample Based Format to a Long Interval Range Format

Description

Legacy function that will not be supported in future updates.

Usage

```r
sample_to_range(data, range = c(0, 50, 90), type = 7, keep_quantile_col = TRUE)
```

Arguments

- **data**: a data.frame with samples
- **range**: a numeric vector of interval ranges to extract (e.g. `c(0, 50, 90)`)
- **type**: type argument passed down to the quantile function. For more information, see `quantile`
- **keep_quantile_col**: keep quantile column, default is TRUE

Value

- a data.frame in long format

sample_to_range_long

Change Data from a Sample Based Format to a Long Interval Range Format

Description

Transform data from a format that is based on predictive samples to a format based on interval ranges

Usage

```r
sample_to_range_long(
  data,
  range = c(0, 50, 90),
  type = 7,
  keep_quantile_col = TRUE
)
```
Arguments

- **data**: a data.frame with samples
- **range**: a numeric vector of interval ranges to extract (e.g. `c(0, 50, 90)`)
- **type**: type argument passed down to the quantile function. For more information, see `quantile`
- **keep_quantile_col**: keep quantile column, default is TRUE

Value

A data.frame in a long interval range format

Examples

```r
example_data <- scoringutils::integer_example_data
quantile_data <- scoringutils::sample_to_range_long(example_data)
```

---

### score_heatmap

**Create a Heatmap of a Scoring Metric**

Description

This function can be used to create a heatmap of one metric across different groups, e.g. the interval score obtained by several forecasting models in different locations.

Usage

```r
score_heatmap(
  scores,
  y = "model",
  x,
  metric,
  facet_formula = NULL,
  scales = "free_y",
  ncol = NULL,
  facet_wrap_or_grid = "facet_wrap",
  ylab = y,
  xlab = x
)
```
Arguments

scores  
A data.frame of scores based on quantile forecasts as produced by `eval_forecasts`.

y  
The variable from the scores you want to show on the y-Axis. The default for this is "model"

x  
The variable from the scores you want to show on the x-Axis. This could be something like "horizon", or "location"

metric  
the metric that determines the value and colour shown in the tiles of the heatmap

facet_formula  
facetting formula passed down to ggplot. Default is NULL

scales  
scales argument that gets passed down to ggplot. Only necessary if you make use of faceting. Default is "free_y"

ncol  
Number of columns for facet wrap. Only relevant if `facet_formula` is given and `facet_wrap_or_grid` == "facet_wrap"

facet_wrap_or_grid  
Use ggplot2's `facet_wrap` or `facet_grid`? Anything other than "facet_wrap" will be interpreted as `facet_grid`. This only takes effect if `facet_formula` is not NULL

ylab  
Label for the y-axis. Default is the variable name on the y-axis

xlab  
Label for the x-axis. Default is the variable name on the x-axis

Value

A ggplot2 object showing a heatmap of the desired metric

Examples

```r
scores <- scoringutils::eval_forecasts(scoringutils::quantile_example_data,
  summarise_by = c("model", "value_desc", "range"))

scoringutils::score_heatmap(scores, x = "value_desc", metric = "bias")
```

---

**score_table**  
Plot Coloured Score Table

Description

Plots a coloured table of summarised scores obtained using `eval_forecasts`
score_table

Usage

score_table(
  summarised_scores,
  y = NULL,
  select_metrics = NULL,
  facet_formula = NULL,
  ncol = NULL,
  facet_wrap_or_grid = "facet_wrap"
)

Arguments

summarised_scores
  A data.frame of summarised scores as produced by eval_forecasts

y
  the variable to be shown on the y-axis. If NULL (default), all columns that are
  not scoring metrics will be used. Alternatively, you can specify a vector with
  column names, e.g. y = c("model", "location"). These column names will
  be concatenated to create a unique row identifier (e.g. "model1_location1")

select_metrics
  A character vector with the metrics to show. If set to NULL (default), all metrics
  present in summarised_scores will be shown

facet_formula
  formula for facetting in ggplot. If this is NULL (the default), no facetting will
  take place

ncol
  Number of columns for facet wrap. Only relevant if facet_formula is given
  and facet_wrap_or_grid == "facet_wrap"

facet_wrap_or_grid
  Use ggplot2's facet_wrap or facet_grid? Anything other than "facet_wrap"
  will be interpreted as facet_grid. This only takes effect if facet_formula is
  not NULL

Value

A ggplot2 object with a coloured table of summarised scores

Examples

scores <- scoringutils::eval_forecasts(scoringutils::quantile_example_data,
  summarise_by = c("model", "value_desc"))
scoringutils::score_table(scores, y = "model", facet_formula = ~ value_desc,
  ncol = 1)

# can also put target description on the y-axis
scoringutils::score_table(scores, y = c("model", "value_desc"))

# yields the same result in this case
scoringutils::score_table(scores)

scores <- scoringutils::eval_forecasts(scoringutils::integer_example_data,
This package is designed to help with assessing the quality of predictions. It provides a collection of proper scoring rules and metrics as well that can be accessed independently or collectively through a higher-level wrapper function.

Predictions can be either probabilistic forecasts (generally predictive samples generated by Markov Chain Monte Carlo procedures), quantile forecasts or point forecasts. The true values can be either continuous, integer, or binary.

A collection of different metrics and scoring rules can be accessed through the function `eval_forecasts`. Given a data.frame of the correct form the function will automatically figure out the type of prediction and true values and return appropriate scoring metrics.

The package also has a lot of default visualisation based on the output created by `eval_forecasts`.

- `score_table`
- `correlation_plot`
- `wis_components`
- `range_plot`
- `score_heatmap`
- `plot_predictions`
- `interval_coverage`
- `quantile_coverage`

Alternatively, the following functions can be accessed directly:

- `brier_score`
- `pit`
- `bias`
- `quantile_bias`
- `sharpness`
- `crps`
- `logs`
- `dss`
Predictions can be evaluated in a lot of different formats. If you want to convert from one format to the other, the following helper functions can do that for you:

- `sample_to_range_long`
- `sample_to_quantile`
- `quantile_to_range_long`
- `range_long_to_quantile`

### sharpness

**Determines sharpness of a probabilistic forecast**

**Description**

Determines sharpness of a probabilistic forecast

**Usage**

```
sharpness(predictions)
```

**Arguments**

- `predictions` nxN matrix of predictive samples, n (number of rows) being the number of data points and N (number of columns) the number of Monte Carlo samples

**Details**

Sharpness is the ability of the model to generate predictions within a narrow range. It is a data-independent measure, and is purely a feature of the forecasts themselves.

Sharpness of predictive samples corresponding to one single true value is measured as the normalised median of the absolute deviation from the median of the predictive samples. For details, see `mad`

**Value**

vector with sharpness values

**References**


**Examples**

```r
predictions <- replicate(200, rpois(n = 30, lambda = 1:30))
sharpness(predictions)
```
**show_avail_forecasts**  
*Visualise Where Forecasts Are Available*

**Description**

Visualise Where Forecasts Are Available

**Usage**

```r
show_avail_forecasts(
    data,
    y = "model",
    x = "forecast_date",
    make_x_factor = TRUE,
    summarise_by = NULL,
    collapse_to_one = TRUE,
    by = NULL,
    show_numbers = TRUE,
    facet_formula = NULL,
    facet_wrap_or_grid = "facet_wrap",
    scales = "fixed",
    legend_position = "none"
)
```

**Arguments**

- `data`  
da.frame with predictions in the same format required for `eval_forecasts`
- `y`  
character vector of length one that denotes the name of the column to appear on the y-axis of the plot
- `x`  
character vector of length one that denotes the name of the column to appear on the x-axis of the plot
- `make_x_factor`  
logical (default is TRUE). Whether or not to convert the variable on the x-axis to a factor. This has an effect e.g. if dates are shown on the x-axis.
- `summarise_by`  
character vector or NULL (the default) that denotes the categories over which the number of forecasts should be summed up. By default (i.e. `summarise_by = NULL`) this will be all the columns that appear in either x, y, or the facetting formula.
- `collapse_to_one`  
logical. If TRUE) (the default), everything not included in by will be counted only once. This is useful, for example, if you don’t want to count every single sample or quantile, but instead treat one set of samples or quantiles as one forecast.
- `by`  
character vector or NULL (the default) that denotes the unit of an individual forecast. This argument behaves similarly to the by argument in link{eval_forecasts}. By default, all columns are used that are not part of any internally protected columns like "sample" or "prediction" or similar. The by argument is only necessary if `collapse_to_one = TRUE` to indicate which rows not to collapse to one.
**show_numbers**

logical (default is TRUE) that indicates whether or not to show the actual count numbers on the plot.

**facet_formula**

formula for facetting in ggplot. If this is NULL (the default), no facetting will take place.

**facet_wrap_or_grid**

character. Use ggplot2’s facet_wrap or facet_grid? Anything other than "facet_wrap" will be interpreted as facet_grid. This only takes effect if facet_formula is not NULL.

**scales**

character. The scales argument gets passed down to ggplot. Only necessary if you make use of facetting. Default is "fixed".

**legend_position**

character that indicates where to put the legend. The argument gets passed to ggplot2. By default ("none"), no legend is shown.

**Value**

ggplot object with a plot of interval coverage

**Examples**

```r
example1 <- scoringutils::range_example_data_long
show_avail_forecasts(example1, x = "value_date", facet_formula = ~ value_desc)
```

---

**update_list**

*Update a List*

**Description**

‘r lifecycle::badge("stable")’ Used to handle updating settings in a list. For example when making changes to ‘interval_score_arguments’ in ‘eval_forecasts()’

**Usage**

```r
update_list(defaults = list(), optional = list())
```

**Arguments**

- **defaults**
  - A list of default settings

- **optional**
  - A list of optional settings to override defaults

**Value**

A list
**Plot Contributions to the Weighted Interval Score**

**Description**

Visualise the components of the weighted interval score: penalties for over-prediction, under-prediction and for a lack of sharpness

**Usage**

```r
wis_components(
  scores,
  x = "model",
  group = NULL,
  relative_contributions = FALSE,
  facet_formula = NULL,
  scales = "free_y",
  ncol = NULL,
  facet_wrap_or_grid = "facet_wrap",
  x_text_angle = 90,
  xlab = x,
  ylab = "WIS contributions"
)
```

**Arguments**

- **scores**  
  A data.frame of scores based on quantile forecasts as produced by `eval_forecasts`

- **x**  
  The variable from the scores you want to show on the x-Axis. Usually this will be "model"

- **group**  
  Choose a grouping variable for the plot that gets directly passed down to ggplot. Default is NULL

- **relative_contributions**  
  Show relative contributions instead of absolute contributions. Default is FALSE and this functionality is not available yet.

- **facet_formula**  
  Facetting formula passed down to ggplot. Default is NULL

- **scales**  
  Scales argument that gets passed down to ggplot. Only necessary if you make use of faceting. Default is "free_y"

- **ncol**  
  Number of columns for facet wrap. Only relevant if facet_formula is given and facet_wrap_or_grid == "facet_wrap"

- **facet_wrap_or_grid**  
  Use ggplot2's `facet_wrap` or `facet_grid`? Anything other than "facet_wrap" will be interpreted as `facet_grid`. This only takes effect if facet_formula is not NULL

- **x_text_angle**  
  Angle for the text on the x-axis. Default is 90

- **xlab**  
  Label for the x-axis. Default is the variable name on the x-axis

- **ylab**  
  Label for the y-axis. Default is "WIS contributions"
Value

A ggplot2 object showing contributions from the three components of the weighted interval score.

References


Examples

```
scores <- scoringutils::eval_forecasts(scoringutils::quantile_example_data,
summarise_by = c("model", "value_desc"))
scoingutils::wis_components(scores, x = "model", facet_formula = ~ value_desc,
relative_contributions = TRUE)
scoingutils::wis_components(scores, x = "model", facet_formula = ~ value_desc,
relative_contributions = FALSE)
```
Index

* datasets
  binary_example_data, 9
  continuous_example_data, 13
  example_quantile_forecasts_only, 24
  example_truth_data_only, 25
  integer_example_data, 28
  quantile_example_data, 45
  range_example_data_long, 48
  range_example_data_semi_wide, 49
  range_example_data_wide, 50

  abs_error, 3
  add_quantiles, 4
  add_rel_skill_to_eval_forecasts, 5
  add_sd, 6
  ae_median_quantile, 6
  ae_median_sample, 7, 61

  bias, 8, 18, 60
  binary_example_data, 9
  brier_score, 10, 18, 60

  check_equal_length, 11
  check_not_null, 11
  compare_two_models, 12, 12, 33–35
  continuous_example_data, 13
  correlation_plot, 13, 60
  crps, 14, 19, 60
  crps_sample, 14

  delete_columns, 15
  dss, 15, 19, 60
  dss_sample, 15

  eval_forecasts, 5, 12, 13, 16, 22, 24, 29, 32, 33, 35, 41, 45, 46, 48, 51–54, 58–60, 62, 64
  eval_forecasts_binary, 20
  eval_forecasts_sample, 22
  example_quantile_forecasts_only, 24
  example_truth_data_only, 25
  extract_from_list, 26
  geom_mean_helper, 26

  hist_PIT, 27
  hist_PIT_quantile, 27
  integer_example_data, 28
  interval_coverage, 28, 60
  interval_score, 18, 29

  logs, 19, 31, 60
  logs_sample, 31

  mad, 61
  merge_pred_and_obs, 32
  mse, 32

  pairwise_comparison, 5, 12, 18, 33, 34, 40
  pairwise_comparison_one_group, 12, 34, 35
  pit, 18, 24, 35, 38, 39, 60
  pit_df, 38
  pit_df_fast, 39
  plot_pairwise_comparison, 40
  plot_predictions, 41, 60

  quantile, 55–57
  quantile_bias, 43, 60
  quantile_coverage, 44, 60
  quantile_example_data, 45, 47
  quantile_to_long, 46
  quantile_to_range, 47
  quantile_to_range_long, 17, 21, 23, 47, 61
  quantile_to_wide, 48

  range_example_data_long, 46–48, 48, 51, 52, 54
  range_example_data_semi_wide, 49
  range_example_data_wide, 50, 52, 54
range_long_to_quantile, 17, 21, 23, 51, 61
range_long_to_wide, 51, 52
range_plot, 52, 60
range_to_quantile, 54
range_wide_to_long, 54

sample_to_quantile, 17, 21, 23, 55, 61
sample_to_range, 56
sample_to_range_long, 17, 21, 23, 56, 61
score_heatmap, 57, 60
score_table, 58, 60
scoringutils, 60
sharpness, 18, 60, 61
show_avail_forecasts, 62

update_list, 63

wis_components, 60, 64