Evaluating Forecasts with scoringutils in R

Nikos I. Bosse
London School of Hygiene & Tropical Medicine (LSHTM)

Hugo Gruson
LSHTM

Anne Cori
Imperial College London

Edwin van Leeuwen
UK Health Security Agency, LSHTM

Sebastian Funk
LSHTM

Sam Abbott
LSHTM

Abstract

Evaluating forecasts is essential to understand and improve forecasting and make forecasts useful to decision makers. A variety of R packages provide a broad variety of scoring rules, visualisations and diagnostic tools. One particular challenge, which **scoringutils** aims to address, is handling the complexity of evaluating and comparing forecasts from several forecasters across multiple dimensions such as time, space, and different types of targets. **scoringutils** extends the existing landscape by offering a convenient and flexible **data.table**-based framework for evaluating and comparing probabilistic forecasts (forecasts represented by a full predictive distribution). Notably, **scoringutils** is the first package to offer extensive support for probabilistic forecasts in the form of predictive quantiles, a format that is currently used by several infectious disease Forecast Hubs. The package is easily extendable, meaning that users can supply their own scoring rules or extend existing classes to handle new types of forecasts. **scoringutils** provides broad functionality to check the data and diagnose issues, to visualise forecasts and missing data, to transform data before scoring, to handle missing forecasts, to aggregate scores, and to visualise the results of the evaluation. The paper presents the package and its core functionality and illustrates common workflows using example data of forecasts for COVID-19 cases and deaths submitted to the European COVID-19 Forecast Hub.

Keywords: forecasting, forecast evaluation, proper scoring rules, scoring, R.

1. Introduction

Good forecasts are of great interest to decision makers in various fields like finance (Timmermann 2018; Elliott and Timmermann 2016), weather predictions (Gneiting and Raftery 2005; Kukkonen *et al.* 2012) or infectious disease modeling (Reich *et al.* 2019; Funk *et al.* 2020; Cramer *et al.* 2021; Bracher *et al.* 2022; Sherratt *et al.* 2022). For decades, researchers, especially in the field of weather forecasting, have therefore developed and refined an arsenal of techniques to evaluate predictions (see for example Good (1952), Epstein (1969); Murphy (1971); Matheson and Winkler (1976), Gneiting, Balabdaoui, and Raftery (2007), Funk, Camacho, Kucharski, Lowe, Eggo, and Edmunds (2019), Gneiting and Raftery (2007), Bracher,

Ray, Gneiting, and Reich (2021)).

Various R (R Core Team 2021) packages cover a wide variety of scoring rules, plots and metrics that are useful in assessing the quality of a forecast. Existing packages offer functionality that is well suited to evaluate a variety of predictive tasks, but also come with important limitations.

Some packages such as **tscount** (Liboschik, Fokianos, and Fried 2017), **topmodels** (Zeileis and Lang 2022), **GLMMadaptive** (Rizopoulos 2023), **cvGEE** (Rizopoulos 2019) or **fabletools** (O'Hara-Wild, Hyndman, and Wang 2023) expect that forecasts were generated in a certain way and require users to supply an object of a specific class to compute scores. These packages provide excellent tools for users operating within the specific package framework but are by their nature not generally applicable to many use cases practitioners might encounter.

Packages such as **scoringRules** (Jordan, Krüger, and Lerch 2019), **Metrics** (Hamner and Frasco 2018), **MLmetrics** (Yan 2016), **verification** (Laboratory 2015), **SpecsVerification** (Siegert 2020), **surveillance** (Meyer, Held, and Höhle 2017), **predtools** (Sadatsafavi, Safari, and Lee 2023), or **probably** (Kuhn, Vaughan, and Ruiz 2023b) provide an extensive collection of tools, scoring rules and visualisations for various use cases. However, most scoring functions operate on vectors and matrices. This is desirable in many applications but can make it difficult to simultaneously evaluate multiple forecasts across several dimensions, such as time, space, and different types of targets.

scoring (Merkle and Steyvers 2013) operates on a data.frame and uses a formula interface, making this task easier. However, **scoring** only exports a few scoring rules and does not allow users to supply their own. **yardstick** (Kuhn, Vaughan, and Hvitfeldt 2023a), which builds on the **tidymodels** (Kuhn and Wickham 2020) framework, is the most general and flexible other forecast evaluation package. It allows users to apply arbitrary scoring rules to a data.frame of forecasts, independently of how they were created. However, **yardstick** is primarily focused on point forecasts and classification tasks. It currently lacks general support for probabilistic forecasts (forecasts in the form of a full predictive distribution, represented e.g. by a set of quantiles or samples from the forecast distribution). Probabilistic forecasts are desirable, as they allow decision makers to take into account the uncertainty of a forecast (Gneiting *et al.* 2007), and are widely used, e.g. in Meteorology or Epidemiology.

scoringutils aims to fill the existing gap in the ecosystem by providing a flexible general-purpose tool for the evaluation of probabilistic forecasts. It offers a coherent `data.table`-based framework and workflow that allows users to evaluate and compare forecasts across multiple dimensions using a wide variety of default and user-provided scoring rules. Notably, **scoringutils** is the first package to offer extensive support for probabilistic forecasts in the form of predictive quantiles, a format that is currently used by several infectious disease Forecast Hubs (Reich *et al.* 2019; Cramer *et al.* 2020; Sherratt *et al.* 2022; Bracher *et al.* 2022). The package provides broad functionality to check the data and diagnose issues, to visualise forecasts and missing data, to transform data before scoring (see Bosse, Abbott, Cori, van Leeuwen, Bracher, and Funk 2023), to apply various metrics and scoring rules to data, to handle missing forecasts, to aggregate scores and to visualise the results of the evaluation. **scoringutils** makes extensive use of **data.table** (Dowle and Srinivasan 2023) to ensure fast and memory-efficient computations. The core functionality is designed around S3 classes, allowing users to expand on the generics and methods implemented in the package. **scoringutils** provides extensive documentation and case studies, as well as sensible defaults

for scoring forecasts.

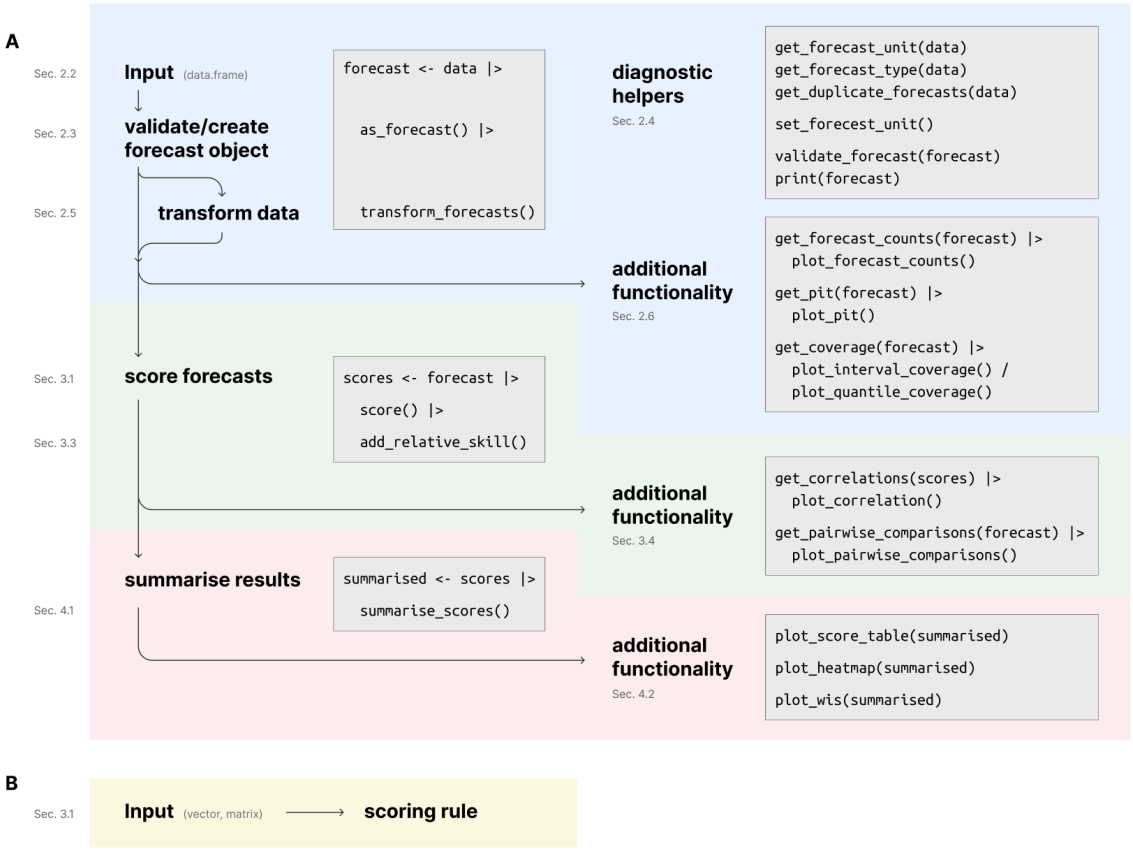

Figure 1: Illustration of the suggested workflow for evaluating forecasts with **scoringutils**. A: Workflow for working with forecasts in a `data.table`-based format. The left side shows the core workflow of the package: 1) validating and processing inputs, 2) scoring forecasts and 3) summarising scores. The right side shows additional functionality that is available at the different stages of the evaluation process. The part in blue is covered by Section 2 and includes all functions related to processing and validating inputs as well as obtaining additional information about the forecasts. The part in green is covered by Section 3 and includes all functions related to scoring forecasts and obtaining additional information about the scores. The part in red is covered by Section 4 and includes all functions related to summarising scores and additional visualisations based on summarised scores. B: An alternative workflow, allowing users to call scoring rules directly with vectors/matrices as inputs.

Paper outline and package workflow

The structure of this paper follows the suggested package workflow which consists of 1) validating and processing inputs, 2) scoring forecasts and 3) summarising scores. This workflow is illustrated in Figure 1, which displays the core workflow (left side) as well as additional functionality that is available at different stages of the evaluation process (right side).

Section 2 is centred around validating inputs, `forecast` objects, and the associated function-

ality. It explains the expected input formats and how to validate inputs and diagnose issues. It provides an overview of the types of forecasts supported by **scoringutils** and the different S3 classes used to represent these forecast types. It also provides information on a variety of functions that can be used to visualise forecasts, transform inputs or obtain additional information and visualisations.

Section 3 is centred around scoring forecasts and the additional functionality that is available to manipulate and analyse scores further. It explains how to score forecasts, either in a `data.table`-format or in a format based on matrices and vectors. It also provides information on additional information that can be computed from scores, such as correlations between scores or relative skill scores based on pairwise comparisons. These can be useful to mitigate the effects of missing forecasts.

Section 4 is centred around summarised scores. It explains how to summarise scores and gives information on additional visualisations that can be created based on summarised scores.

Section 5 discusses the merits and limitations of the package in its current version as explores avenues for future work.

All functionality will be illustrated using the example data shipped with the package, which is based on a subset of case and death forecasts submitted every week between May and September 2021 to the European COVID-19 Forecast Hub (Sherratt *et al.* 2022). Following the convention of the different COVID-19 Forecast Hubs, we will restrict examples to two-week-ahead forecasts.

The code for this package and paper can be found on `https:github.com/epiforecasts/scoringutils`. The full package documentation as well as an overview of all existing functions can also be seen on `https://epiforecasts.io/scoringutils`.

2. Inputs, forecast types and input validation

2.1. Input formats and types of forecasts

Forecasts differ in the exact prediction task and in how the forecaster chooses to represent their prediction. To distinguish different kinds of forecasts, we use the term "forecast type" (which is more a convenient classification than a formal definition). Currently, `scoringutils` distinguishes four different forecast types: "binary", "point", "quantile" and "sample" forecasts.

- "Binary" denotes a probability forecast for a binary (yes/no) outcome variable. This is sometimes also called "soft binary classification".
- "Point" denotes a forecast for a continuous or discrete outcome variable that is represented by a single number.
- "Quantile" or "quantile-based" is used to denote a probabilistic forecast for a continuous or discrete outcome variable, with the forecast distribution represented by a set of predictive quantiles. While a single quantile would already satisfy the requirements for a quantile-based forecast, most scoring rules expect a set of quantiles which are symmetric around the median (thus forming the lower and upper bounds of central "prediction intervals") and will return `NA` if this is not the case.

- "Sample" or "sample-based" is used to denote a probabilistic forecast for a continuous or discrete outcome variable, with the forecast represented by a finite set of samples drawn from the predictive distribution. A single sample technically suffices, but would lead to very imprecise results.

Forecast type			**column**	**type**

All forecast types			`observed`
			`predicted`
			`model`	

Classification	Binary	Soft classification (prediction is probability)	`observed` `predicted`	factor with 2 levels numeric [0,1]
Point forecast			`observed` `predicted`	numeric numeric

Probabilistic forecast		Sample format	`observed`
			`predicted`
			`sample_id`
				numeric
				numeric
				numeric

		Quantile format	`observed`
			`predicted`
			`quantile_level`
				numeric
				numeric
				numeric [0,1]

Table 1: Formatting requirements for data inputs. Regardless of the forecast type, the `data.frame` (or similar) must have columns called `observed`, `predicted`, and `model`. For binary forecasts, the column `observed` must be of type factor with two levels and the column `predicted` must be a numeric between 0 and 1. For all other forecast types, both `observed` and `predicted` must be of type numeric. Forecasts in a sample-based format require an additional numeric column `sample_id` and forecasts in a quantile-based format require an additional numeric column `quantile_level` with values between 0 and 1.

The starting point for working with **scoringutils** is usually a `data.frame` (or similar) containing both the predictions and the observed values. In a next step (see Section 2.2) this data will be validated and transformed into a "forecast object". The input data needs to have a column `observed` for the observed values, a column `predicted` for the predicted values, and a column `model` denoting the name of the model/forecaster that generated the forecast. Additional requirements depend on the forecast type. Table 1 shows the expected input format for each forecast type.

The package contains example data for each forecast type, which can serve as an orientation for the correct formats. The example data sets are exported as `example_quantile`, `example_continuous`, `example_integer`, `example_point` and `example_binary`. For illustrative purposes, the example data also contains some rows with only observations and no corresponding predictions. Input formats for the scoring rules that can be called directly follow the same convention, with inputs expected to be vectors or matrices.

The unit of a single forecast

Apart from the columns `observed`, `predicted`, `model`, and the extra columns required for each forecast type, it is usually necessary that the input data contains additional columns.

This is because a single probabilistic forecast (apart from binary predictions) is composed of multiple values. A quantile-based forecast, for example, is composed of several quantiles, and a sample-based forecast of multiple samples. However, every row only holds a single sample/quantile. Several rows in the input data therefore jointly form a single forecast. Additional columns in the input provide the information necessary to group rows that belong to the same forecast. The combination of values in those columns forms the unit of a single forecast (or "forecast unit") and should uniquely identify a single forecast. For example, consider forecasts made by different models in various locations at different time points and for different targets. A single forecast could then be uniquely described by the values in the columns `model`, `location`, `date`, and `target`, and the forecast unit would be `forecast_unit = c("model", "location", "date", "target")`.

Rows are automatically grouped based on the values in all other columns present in the data (excluding required columns like `sample_id` or `quantile_level` and values computed by **scoringutils**). As the forecast unit is determined based on all existing columns, no column must be present that is unrelated to the forecast unit. As a very simplistic example, consider an additional row, `"even"`, that is one if the row number is even and zero otherwise. The existence of this column would change results, as **scoringutils** assumes it was relevant to grouping the forecasts.

2.2. Forecast objects and input validation

The raw input data needs to be processed and validated using the function `as_forecast()`:

```
R> library(scoringutils)
R> forecast_quantile <- example_quantile[horizon == 2] |>
+   as_forecast()
```

The function `as_forecast()` recognises the type of the forecast based on the available columns, transforms the input into a "forecast" object and validates it (see Figure A.11 for details). A forecast object is a `data.table` that has passed some input validations. It behaves like a `data.table`, but has dedicated methods e.g. for input validation, scoring and printing. The classes corresponding to the forecast types are `forecast_point`, `forecast_binary`, `forecast_quantile` and `forecast_sample`.

`as_forecast()` can automatically determine the forecast type and forecast unit based on the input data. However, it can also take additional arguments that help facilitate the process of creating a forecast object:

```
R> forecast_quantile <- example_quantile[horizon == 2] |>
+   as_forecast(
+     forecast_unit = c(
+       "model", "location", "target_end_date",
+       "forecast_date", "horizon", "location"
+     ),
+     forecast_type = "quantile",
+     observed = "observed",
+     predicted = "predicted",
```

```
+     model = "model",
+     quantile_level = "quantile_level",
+   )
```

The argument `forecast_unit` allows the user to manually set the unit of a single forecast. This is done by dropping all columns that are not either specified in the `forecast_unit` or are "protected" columns (such as `observed`, `predicted`, `model`, `quantile_level`, or `sample_id`). The argument `forecast_type` allows users to manually specify the forecast type they expect. If the forecast type inferred from the input does not match the specified forecast type, an error is thrown. The other arguments can be used to specify the column names of the input data that correspond to the required columns. `as_forecast()` will rename the specified columns to the corresponding required columns.

2.3. Diagnostic helper functions

Various helper functions are available to diagnose and fix issues with the input data. The most important one is `print()`. Once a forecast object has successfully been created, diagnostic information will automatically be added to the output when printing a forecast object. This information includes the forecast type, the forecast unit, and additional information in case the object fails validations.

```
R> print(forecast_quantile, 2)

Forecast type:
[1] "quantile"

Forecast unit:
[1] "location"        "target_end_date" "target_type"
[4] "location_name"   "forecast_date"   "model"
[7] "horizon"

Key: <location, target_end_date, target_type>
       location target_end_date target_type observed location_name
         <char>          <Date>      <char>    <num>        <char>
    1:       DE      2021-01-02       Cases   127300       Germany
    2:       DE      2021-01-02      Deaths     4534       Germany
   ---
20544:       IT      2021-07-24      Deaths       78         Italy
20545:       IT      2021-07-24      Deaths       78         Italy
       forecast_date quantile_level predicted                model
              <Date>          <num>     <int>               <char>
    1:          <NA>             NA        NA                 <NA>
    2:          <NA>             NA        NA                 <NA>
   ---
20544:    2021-07-12          0.975       611 epiforecasts-EpiNow2
20545:    2021-07-12          0.990       719 epiforecasts-EpiNow2
       horizon
```

```
        <num>
    1:      NA
    2:      NA
   ---
20544:       2
20545:       2
```

Internally, the print method calls the functions `get_forecast_type()`, `get_forecast_unit()` and `validate_forecast()`. `get_forecast_type()` and `get_forecast_unit()` work on either an unvalidated `data.frame` (or similar) or on an already validated forecast object. They return the forecast type and the forecast unit, respectively, as inferred from the input data. `validate_forecast()` re-validates an existing forecast object an can be used programmatically without printing an object (users could in principle also call `as_forecast()` again).

One common issue that causes `as_forecast()` to fail are "duplicates" in the data. **scoringutils** strictly requires that there be only one forecast per forecast unit and only one predicted value per quantile level or sample id within a single forecast. Duplicates usually occur if the forecast unit is misspecified. For example, if we removed the column `target_type` from the example data, we would now have two forecasts (one for cases and one for deaths of COVID-19) that appear to have the same forecast unit (since the information that distinguished between case and death forecasts is no longer there). The function `get_duplicate_forecasts()` returns duplicate rows for the user to inspect. To remedy the issue, the user needs to add additional columns that uniquely identify a single forecast.

```
R> rbind(example_quantile, example_quantile[1001:1002]) |>
+   get_duplicate_forecasts()
```

```
   location target_end_date target_type observed location_name
     <char>          <Date>      <char>    <num>        <char>
1:       DE      2021-05-22      Deaths     1285       Germany
2:       DE      2021-05-22      Deaths     1285       Germany
3:       DE      2021-05-22      Deaths     1285       Germany
4:       DE      2021-05-22      Deaths     1285       Germany
   forecast_date quantile_level predicted                model
          <Date>          <num>     <int>               <char>
1:    2021-05-17          0.975      1642 epiforecasts-EpiNow2
2:    2021-05-17          0.990      1951 epiforecasts-EpiNow2
3:    2021-05-17          0.975      1642 epiforecasts-EpiNow2
4:    2021-05-17          0.990      1951 epiforecasts-EpiNow2
   horizon
     <num>
1:       1
2:       1
3:       1
4:       1
```

2.4. Transforming forecasts

As suggested in Bosse *et al.* (2023), users may want to transform forecasts before scoring them. Two commonly used scoring rules are the continuous ranked probability score (CRPS) and the weighted interval score (WIS). Both measure the absolute distance between the forecast and the observation. This may not be desirable, for example in the context of epidemiological forecasts, where infectious disease processes are usually modelled to occur on a multiplicative scale. Taking the logarithm of the forecasts and observations before scoring them makes it possible to evaluate forecasters based on how well they predicted the exponential growth rate.

The function `transform_forecasts()` takes a validated forecast object as input and allows users to apply arbitrary transformations to forecasts and observations. Users can specify a function via the argument `fun` (as well as supply additional function parameters). The default function is `log_shift()`, which is simply a wrapper around `log()` with an additional argument that allows adding an offset (i.e. `log(x + offset)`) to deal with zeroes in the data. Users can specify to either append the transformed forecasts to the existing data by setting `append = TRUE` (the default behaviour, resulting in an additional column `scale`) or to replace the existing forecasts in place.

The example data contains negative values which need to be handled before applying the logarithm. Presumably, negative values for count data should be dropped altogether, but for illustrative purposes, we will call `transform_forecasts()` twice to replace them with zeroes first before appending transformed counts.

```
R> forecast_quantile |>
+   transform_forecasts(fun = \(x) {pmax(x, 0)}, append = FALSE) |>
+   transform_forecasts(fun = log_shift, offset = 1) |>
+   print(2)
```

```
       location target_end_date target_type     observed
         <char>          <Date>      <char>        <num>
    1:       DE      2021-01-02       Cases 1.273000e+05
    2:       DE      2021-01-02      Deaths 4.534000e+03
   ---
41089:       IT      2021-07-24      Deaths 4.369448e+00
41090:       IT      2021-07-24      Deaths 4.369448e+00
       location_name forecast_date quantile_level predicted
              <char>        <Date>          <num>     <num>
    1:       Germany          <NA>             NA        NA
    2:       Germany          <NA>             NA        NA
   ---
41089:         Italy    2021-07-12          0.975  6.416732
41090:         Italy    2021-07-12          0.990  6.579251
                     model horizon   scale
                    <char>   <num>  <char>
    1:                <NA>      NA natural
    2:                <NA>      NA natural
   ---
41089: epiforecasts-EpiNow2       2     log
41090: epiforecasts-EpiNow2       2     log
```

2.5. Additional functionality related to forecast objects

scoringutils offers a variety of different functions that allow users to obtain and visualise additional information about their forecast. The package also has an extensive Vignette with examples for further visualisations that are not implemented as functions.

Displaying the number of forecasts available

Users can get an overview of how many forecasts there are using `get_forecast_counts()`. The function takes a validated forecast object as input and returns a data.table of forecast counts, which helps obtain an overview of missing forecasts. This can impact the evaluation, if missingness correlates with performance. Users can specify the level of summary through the `by` argument. For example, to see how many forecasts there are per `model`, `target_type` and `forecast_date`, we can run

```
R> forecast_counts <- forecast_quantile |>
+   get_forecast_counts(
+     by = c("model", "target_type", "forecast_date")
+   )
```

We can visualise the results by calling `plot_forecast_counts()` on the output (Figure 2).

```
R> library(ggplot2)
R> forecast_counts |>
+   plot_forecast_counts(x = "forecast_date") +
+   facet_wrap(~ target_type) +
+   labs(y = "Model", x = "Forecast date")
```

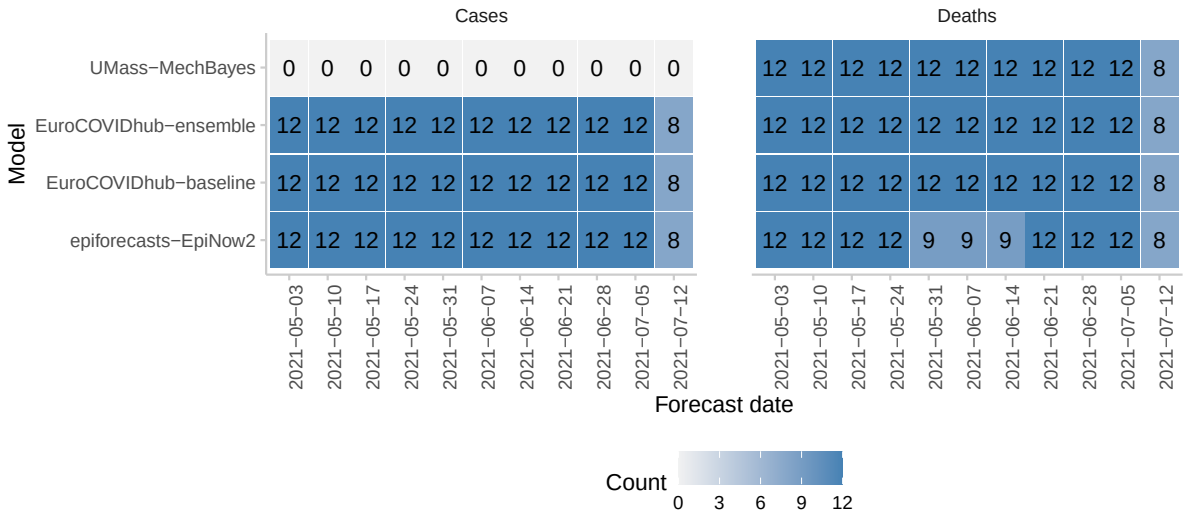

Figure 2: Visualistion of forecast counts for the example data. Numbers (and colour shade) indicate the number of forecasts available for a given model, target type and forecast date.

Probabilistic calibration and PIT histograms

One important quality of good forecasts is calibration. The term describes a statistical consistency between the forecasts and the observations, i.e. an absence of systematic deviations between the two. It is possible to distinguish several forms of calibration which are discussed in detail by Gneiting *et al.* (2007). The form of calibration most commonly focused on is called probabilistic calibration. Probabilistic calibration means that the forecast distributions are consistent with the true data-generating distributions in the sense that on average, τ% of true observations will be below the corresponding τ-%-quantiles of the cumulative forecast distributions.

A common way to visualise probabilistic calibration is the probability integral transform (PIT) histogram (Dawid 1984). Observed values, y, are transformed using the CDF of the predictive distribution, F, to create a new variable u with $u = F(y)$. u is therefore simply the CDF of the predictive distribution evaluated at the observed value. If forecasts are probabilistically calibrated, then the transformed values will be uniformly distributed (for a proof see for example Angus (1994)). When plotting a histogram of PIT values (see Figure 3), a systematic bias usually leads to a triangular shape, a U-shaped histogram corresponds to forecasts that are underdispersed (too sharp) and a hump shape appears when forecasts are overdispersed (too wide). There exist different variations of the PIT to deal with discrete instead of continuous data (see e.g. Czado, Gneiting, and Held (2009) and Funk *et al.* (2019)). The PIT version implemented in `scoringutils` for discrete variables follows Funk *et al.* (2019).

Users can obtain PIT histograms based on validated forecast objects using the function `get_pit()` and can visualise results using `plot_pit()`. Once again, the argument `by` controls the summary level. The output of the following is shown in Figure 3:

```
R> example_continuous |>
+   get_pit(by = c("model", "target_type")) |>
+   plot_pit() +
+   facet_grid(target_type ~ model)
```

It is, in theory, possible to conduct a formal test for probabilistic calibration, for example by employing an Anderson-Darling test on the uniformity of PIT values. In practice, this can be difficult as forecasts, and therefore PIT values as well, are often correlated. Personal experience suggests that the Anderson-Darling test is often too quick to reject the null hypothesis of uniformity. It is also important to note that uniformity of the PIT histogram does not guarantee that forecasts are indeed calibrated. Gneiting *et al.* (2007); Hamill (2001) provide examples with different forecasters who are mis-calibrated, but have uniform PIT histograms.

Probabilistic calibration and coverage plots

For forecasts in a quantile-based format, there exists a second way to assess probabilistic calibration: we can easily compare the proportion of observations that fall below the τ-quantiles of all forecasts ("empirical quantile coverage") to the nominal quantile coverage τ. Similarly, we can compare the empirical coverage of the central prediction intervals formed by the predictive quantiles to the nominal interval coverage. For example, the central 50% prediction intervals of all forecasts should contain around 50% of the observed values, the 90% central intervals should contain around 90% of observations etc. In addition, we can define coverage deviation as the difference between nominal and empirical coverage.

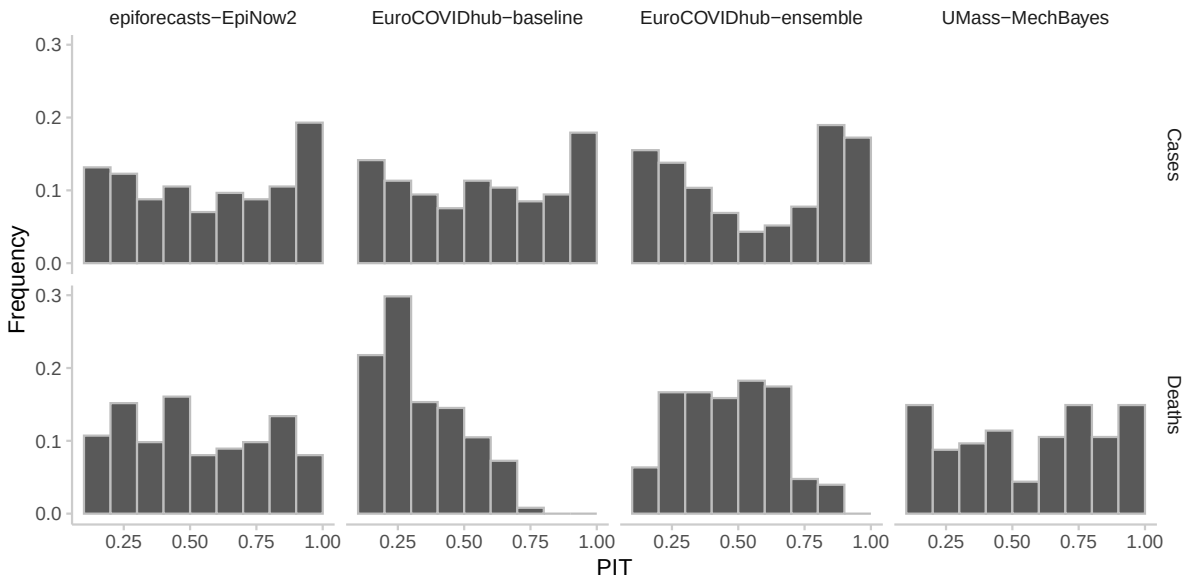

Figure 3: PIT histograms of all models stratified by forecast target. Histograms should ideally be uniform. A u-shape usually indicates overconfidence (forecasts are too narrow), a hump-shaped form indicates underconfidence (forecasts are too uncertain) and a triangle-shape indicates bias.

Interval and quantile coverage can easily be computed by calling `get_coverage()` on a validated forecast object (in a quantile-based format). The function computes interval coverage, quantile coverage, interval coverage deviation and quantile coverage deviation and returns a `data.table` with corresponding columns. Coverage values will be summarised according to the level specified in the `by` argument and one value per quantile level/interval range is returned.

```
R> forecast_quantile |>
+   get_coverage(by = "model") |>
+   print(2)
```

Results can then be visualised using the functions `plot_interval_coverage()` (see Figure 4A) and `plot_quantile_coverage()` (see 4B). Both show nominal against empirical coverage. Ideally, forecasters should lie on the diagonal line. If the line moves into the green-shaded area, the forecaster is too conservative, i.e. the predictive distributions are too wide/overdispersed on average. The white area implies overconfidence/predictive distributions that are too narrow on average (see Figure B.12) for more details).

```
R> coverage <- get_coverage(forecast_quantile, by = c("model", "target_type"))
R>
R> plot_interval_coverage(coverage) +
+   facet_wrap(~ target_type)
R>
R> plot_quantile_coverage(coverage) +
+   facet_wrap(~ target_type)
```

Note that users can also compute individual coverage values as scores using `score()`. This represents a separate workflow that allows users to obtain coverage values as a summary measure to be computed alongside other scores, rather than providing a way to visually assess calibration.

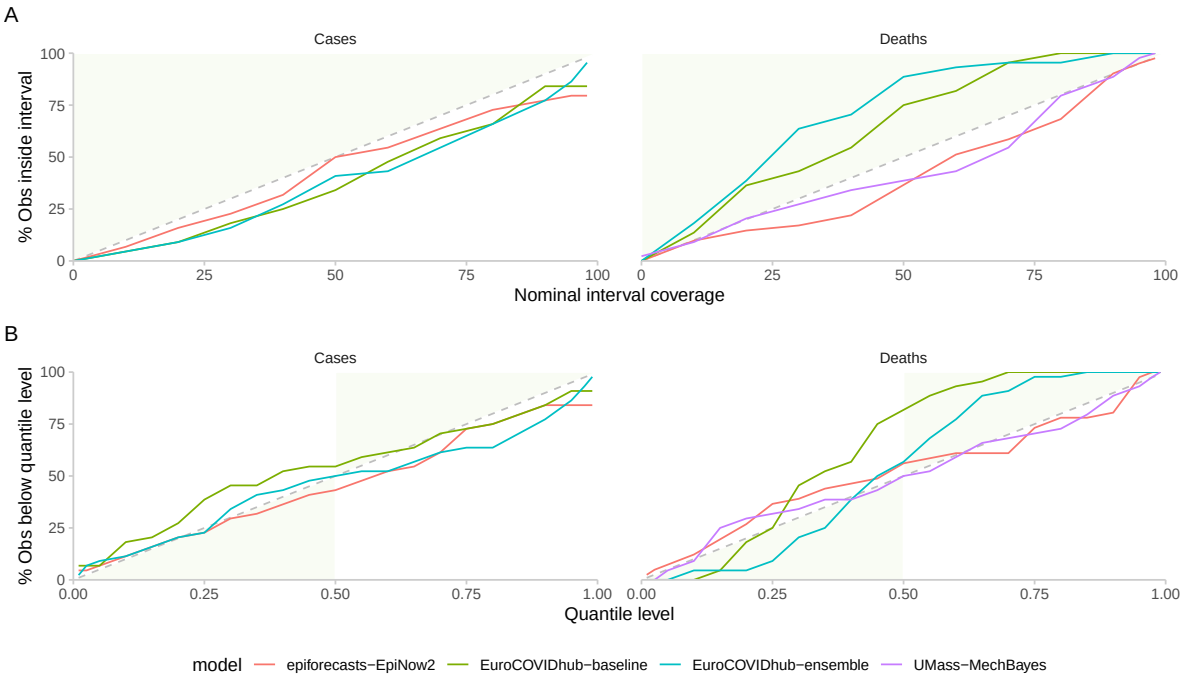

Figure 4: Interval coverage (A) and quantile coverage (B) plots. Areas shaded in green indicate that the forecasts are too wide (i.e., underconfident), while areas in white indicate that the model is overconfident and generates too narrow prediction intervals.

3. Scoring forecasts

Metrics and scoring rules can be applied to data in two different ways: They can be conveniently applied to a data set of observed and predicted values using `score()`, or they be called directly on a set of vectors and matrices. This section will mostly focus on `score()`.

3.1. score() and working with scoring rules

The function `score()` is the workhorse of the package and applies a set of metrics and scoring rules to predicted and observed values. It is a generic function that dispatches to different methods depending on the class of the input. The input of `score()` is a validated forecast object and its output is an object of class `scores`, which is a essentially `data.table` with an additional attribute `metrics` (containing the names of the metrics used for scoring).

```
R> example_point[horizon == 2] |>
+   as_forecast() |>
+   score() |>
+   print(2)
```

```
Key: <location, target_end_date, target_type>
     location target_end_date target_type observed location_name
       <char>          <Date>      <char>    <num>        <char>
  1:       DE      2021-05-15       Cases    64985       Germany
  2:       DE      2021-05-15       Cases    64985       Germany
 ---
304:       IT      2021-07-24      Deaths       78         Italy
305:       IT      2021-07-24      Deaths       78         Italy
     forecast_date predicted                 model horizon ae_point
            <Date>     <int>                <char>   <num>    <num>
  1:    2021-05-03    110716 EuroCOVIDhub-ensemble       2    45731
  2:    2021-05-03    132607 EuroCOVIDhub-baseline       2    67622
 ---
304:    2021-07-12       124       UMass-MechBayes       2       46
305:    2021-07-12       186  epiforecasts-EpiNow2       2      108
       se_point       ape
          <num>     <num>
  1: 2091324361 0.7037162
  2: 4572734884 1.0405786
 ---
304:       2116 0.5897436
305:      11664 1.3846154
```

All `score()` methods take an argument `metrics` with a named list of functions to apply to the data. These can be metrics exported by **scoringutils** or any other custom scoring function. All metrics scoring rules passed to `score()` need to adhere to the same input format (see Figure 5), corresponding to the type of forecast to be scored. Scoring functions must accept a vector of observed values as their first argument, a matrix/vector of predicted values as their second argument and, for quantile-based forecasts, a vector of quantile levels as their third argument). However, functions may have arbitrary argument names. Within `score()`, inputs like the observed and predicted values, quantile levels etc. are passed to the individual scoring rules by position, rather than by name. The default scoring rules for point forecasts, for example, comprise functions from the **Metrics** package, which use the names `actual` and `predicted` for their arguments instead of `observed` and `predicted`. Additional arguments can be passed down to the scoring functions via the `...` arguments in `score()`.

Composing a custom list of metrics and scoring rules

For every forecast type, there exists a default list of scoring rules that are applied to the data when calling `score()`. The default lists can be accessed by calling the functions `metrics_point()`, `metrics_binary()`, `metrics_sample()` and `metrics_quantile()`. These functions take additional arguments `exclude` and `select` which can be used to customise which scoring rules are included. Alternatively, users can call the function `select_metrics()` on a list of scoring rules, which achieves the same purposes and allows users to compose custom lists of metrics and scoring rules.

```
R> custom_metrics <- metrics_quantile() |>
+   select_metrics(select = c("wis", "overprediction"))
```

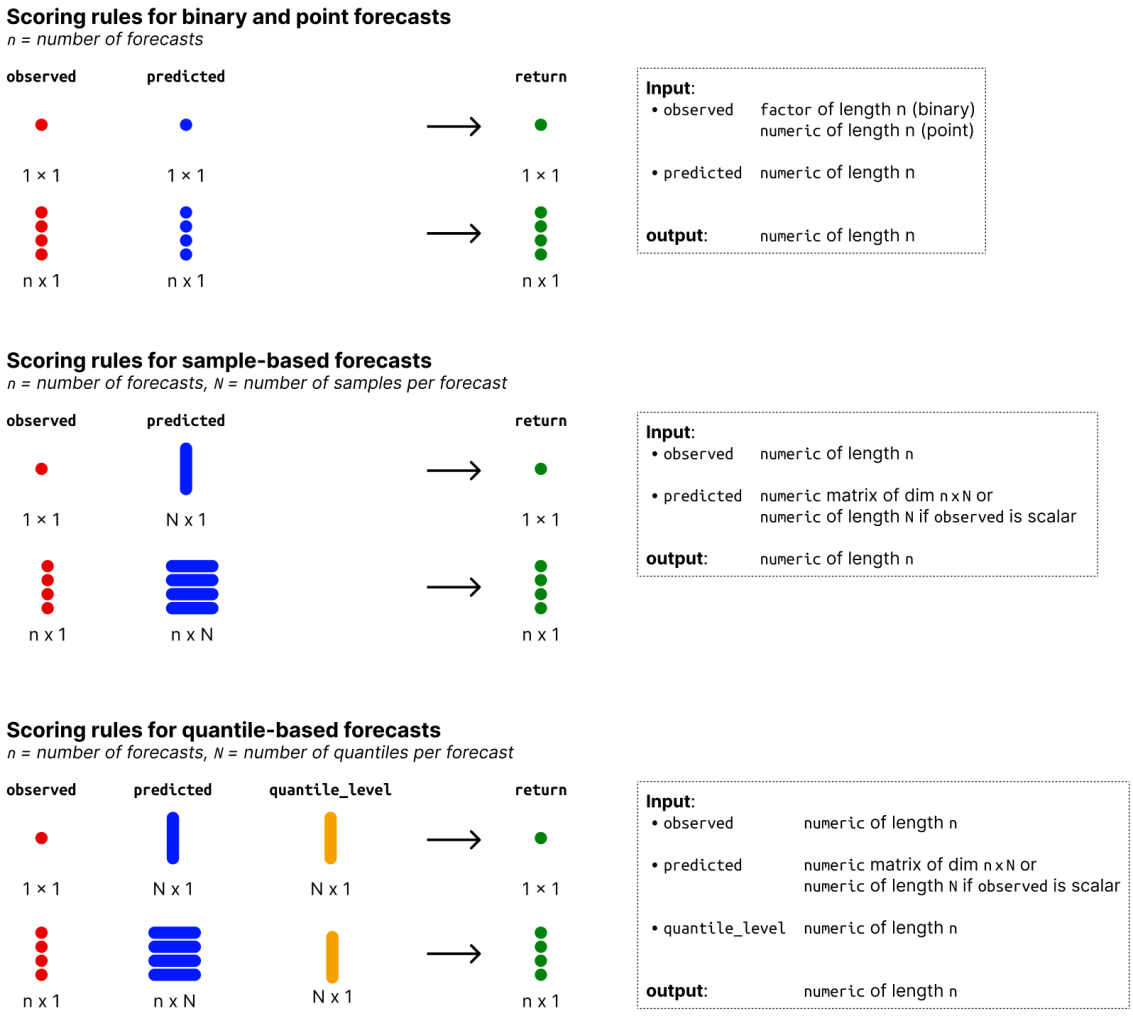

Figure 5: Overview of the inputs and outputs of the metrics and scoring rules exported by **scoringutils**. Dots indicate scalar values, while bars indicate vectors (comprised of values that belong together). Several bars (vectors) can be grouped into a matrix with rows representing the individual forecasts. All scoring functions used within `score()` must accept the same input formats as the functions here. However, functions used within `score()` do not necessarily have to have the same argument names (see Section 3). Input formats directly correspond to the required columns for the different forecast types (see Table 1). The only exception is the forecast type 'sample': Inputs require a column `sample_id` in `score()`, but no corresponding argument is necessary when calling scoring rules directly on vectors or matrices.

```
R>
R> score(metrics = custom_metrics)
```

Details on metrics exported by scoringutils

All metrics are named according to the following schema: `{metric name}_{forecast type}`.

If only a single forecast type is possible, then `_{forecast type}` is omitted. The return value is a vector with scores (only in the case of `wis()`, which is composed of three components (see C), is there an optional argument that causes the function to return a list of vectors for the individual WIS components). The first argument of all metrics exported by **scoringutils** is always `observed`, and the second one is `predicted`. Scoring rules for quantile-based forecasts have an additional argument, `quantile_level`, to denote the quantile levels of the predictive quantiles.

Metrics exported by **scoringutils** differ in the relationship between input and output. Some scoring rules have a one-to-one relationship between predicted values and scores, returning one value per value in `predicted`. This is the case for all metrics for binary and point forecasts. Other scoring rules have a many-to-one relationship, returning one value per multiple values in `predicted`. This is the case for all scoring rules for sample- and quantile-based forecasts. For sample- and quantile-based forecasts, `predicted` is therefore a matrix, with values in each row jointly forming a single forecast.

Input formats and return values are shown in more detail in Figure 5. The package vignettes provide extensive documentation for the metrics exported by **scoringutils** and offer guidance on which scoring rule to use and how to interpret the scores.

3.2. Adding relative skill scores based on pairwise comparisons

Raw scores for different forecasting models are usually not directly comparable when there are missing forecasts in the data set, as missingness is often correlated with predictive performance. One way to mitigate this are relative skill scores based on pairwise comparisons (Cramer *et al.* 2021).

Models enter a 'pairwise tournament', where all possible pairs of models are compared based on the overlapping set of available forecasts common to both models (omitting comparisons where there is no overlapping set of forecasts). For every pair, the ratio of the mean scores of both models is computed. The relative skill score of a model is then the geometric mean of all mean score ratios which involve that model (see Figure 6. This gives us an indicator of performance relative to all other models, with the orientation depending on the score used: if lower values are better for a particular scoring rule, then the same is true for the relative skill score computed based on that score.

Two models can of course only be fairly compared if they have overlapping forecasts. Furthermore, pairwise comparisons between models for a given score are only possible if all values have the same sign, i.e. all score values need to be either positive or negative.

To compute relative skill scores, users can call `add_pairwise_comparison()` on the output of `score()`. This function computes relative skill values with respect to a score specified in the argument `metric` and adds them as an additional column to the input data. Optionally, users can specify a baseline model to also compute relative skill scores scaled with respect to that baseline. Scaled relative skill scores are obtained by simply dividing the relative skill score for every individual model (computed excluding the baseline) by the relative skill score of the baseline model. Pairwise comparisons are computed according to the grouping specified in the argument `by`: internally, the `data.table` with all scores gets split into different `data.table`s according to the values specified in `by` (excluding the column 'model'). Relative scores are then computed for every individual group separately. In the example below we specify `by = c("model", "target_type")`, which means that there is one relative skill score per model,

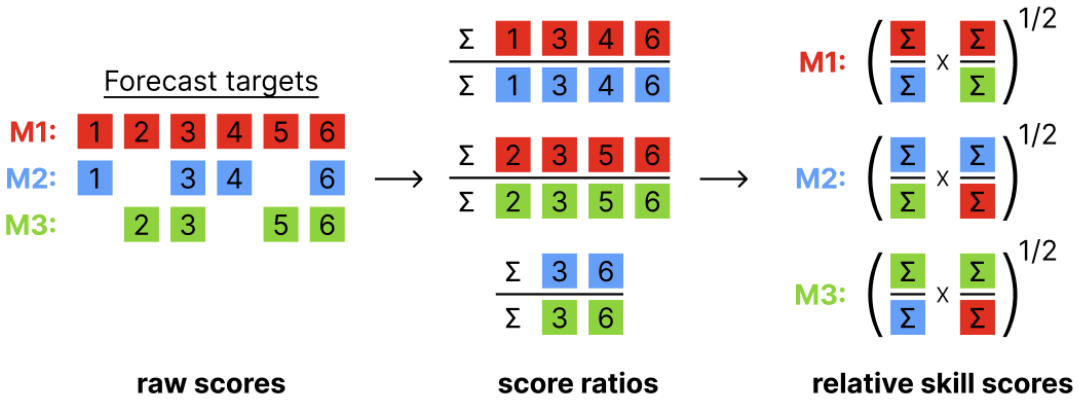

Figure 6: Illustration of the computation of relative skill scores through pairwise comparisons of three different forecast models, M1-M3. Score ratios are computed based on the overlapping set of forecasts common to all pairs of two models. The relative skill score of a model is then the geometric mean of all mean score ratios which involve that model. The orientation of the relative skill score depends on the score used: if lower values are better for a particular scoring rule, then the same is true for the relative skill score computed based on that score.

calculated completely separately for the different forecasting targets.

```
R> forecast_quantile |>
+   score() |>
+   add_relative_skill(by = c("model", "target_type"),
+                      baseline = "EuroCOVIDhub-baseline")
```

Pairwise comparisons should usually be made based on raw, unsummarised scores (meaning that `add_relative_skill()` should be called before `summarise_scores()` (see Section 4)). Summarising scores, for example by computing an average across several dimensions, can change the set of overlapping forecasts between two models and distort relative skill scores.

3.3. Additional functionality related to scores objects

Displaying mean score ratios from pairwise comparisons

scoringutils offers a second alternative workflow to conduct pairwise comparisons between models through the function `get_pairwise_comparisons()`. The purpose of this workflow is to obtain and visualise information on the direct comparisons between every possible pair of models, rather than just computing relative skill scores for every model. The function `get_pairwise_comparisons()` accepts the same inputs as `add_relative_skill()`, and returns a `data.table` with the results of the pairwise tournament. These include the mean score ratios for every pair of models, a p-value for whether scores for one model are significantly different from scores for another model, and the relative and scaled relative skill score for every model (depending on whether a baseline was provided or not).

`get_pairwise_comparisons()` computes p-values using either the Wilcoxon rank sum test (the default, the test is also known as Mann-Whitney-U test) (Mann and Whitney 1947) or a permutation test. P-values are then adjusted using `p.adjust`. In practice, the computation of p-values is complicated by the fact that both tests assume independent observations. In reality, however, forecasts by a model may be correlated across time or space (e.g., if a forecaster has a bad day, they might perform badly across different targets for a given forecast date). P-values may therefore be too liberal in suggesting significant differences where there aren't any. We previously suggested computing relative skill scores based on pairwise comparisons before summarising scores. One exception is the case where one is interested in p-values specifically: One possible way to mitigate issues from correlated forecasts, is to aggregate observations over a category where one suspects correlation (provided there are no missing values within the categories summarised over) to reduce correlation before making pairwise comparisons. A test that is performed on aggregate scores will likely be more conservative.

The mean score ratios resulting from `pairwise_comparison()` can then be visualised using the function `plot_pairwise_comparison()`. An example is shown in Figure 7.

```
R> forecast_quantile |>
+   score() |>
+   get_pairwise_comparisons(by = c("model", "target_type")) |>
+   plot_pairwise_comparisons() +
+   facet_wrap(~ target_type)
```

Correlations between scores

Users can examine correlations between scores using the function `correlations()` and plot the result using `plot_correlations()`. The plot resulting from the following code is shown in Figure 8.

```
R> correlations <- forecast_quantile |>
+   score() |>
+   summarise_scores() |>
+   get_correlations(digits = 2)
R>
R> correlations |>
+   plot_correlations()
```

4. Summarising results

4.1. Summarising scores

Usually, one will not be interested in scores for each individual forecast, but rather in summarised scores. This can be achieved using the function `summarise_scores()`. The function takes a `scores` object (a data.table with an additional attribute `metrics`) as input and applies a summary function to it (by default the mean), returning a `data.table` with summarised scores. Users can set the summary level using the argument `by` and will obtain

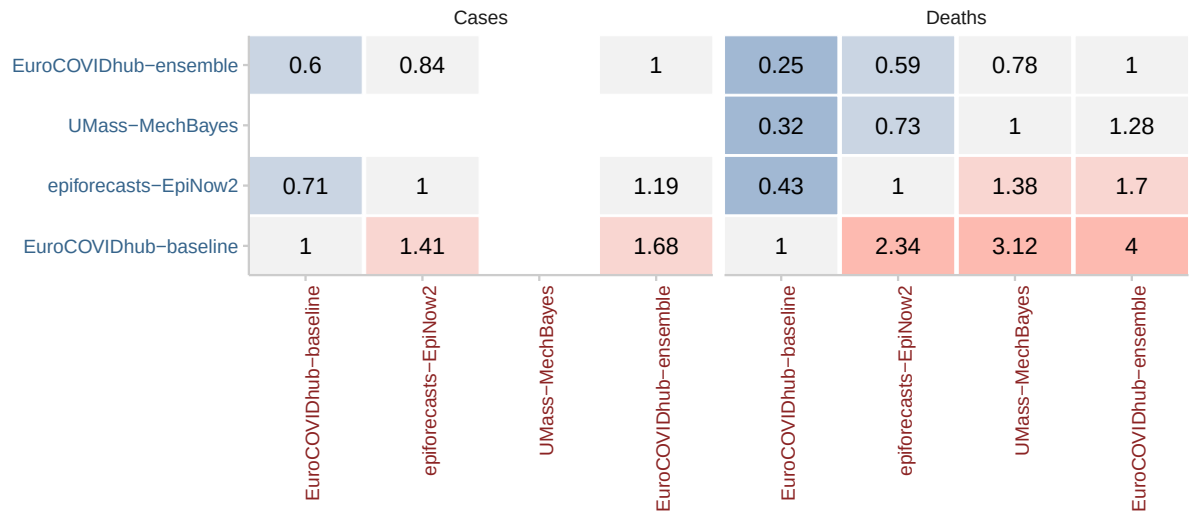

Figure 7: Ratios of mean weighted interval scores based on overlapping forecast sets. When interpreting the plot one should look at the model on the y-axis, and the model on the x-axis is the one it is compared against. If a tile is blue, then the model on the y-axis performed better (assuming that scores are negatively oriented, i.e. that lower scores are better). If it is red, the model on the x-axis performed better in direct comparison. In the example above, the EuroCOVIDhub-ensemble performs best (it only has values smaller than one), while the EuroCOVIDhub-baseline performs worst (and only has values larger than one). For cases, the UMass-MechBayes model is excluded as there are no case forecasts available and therefore the set of overlapping forecasts is empty.

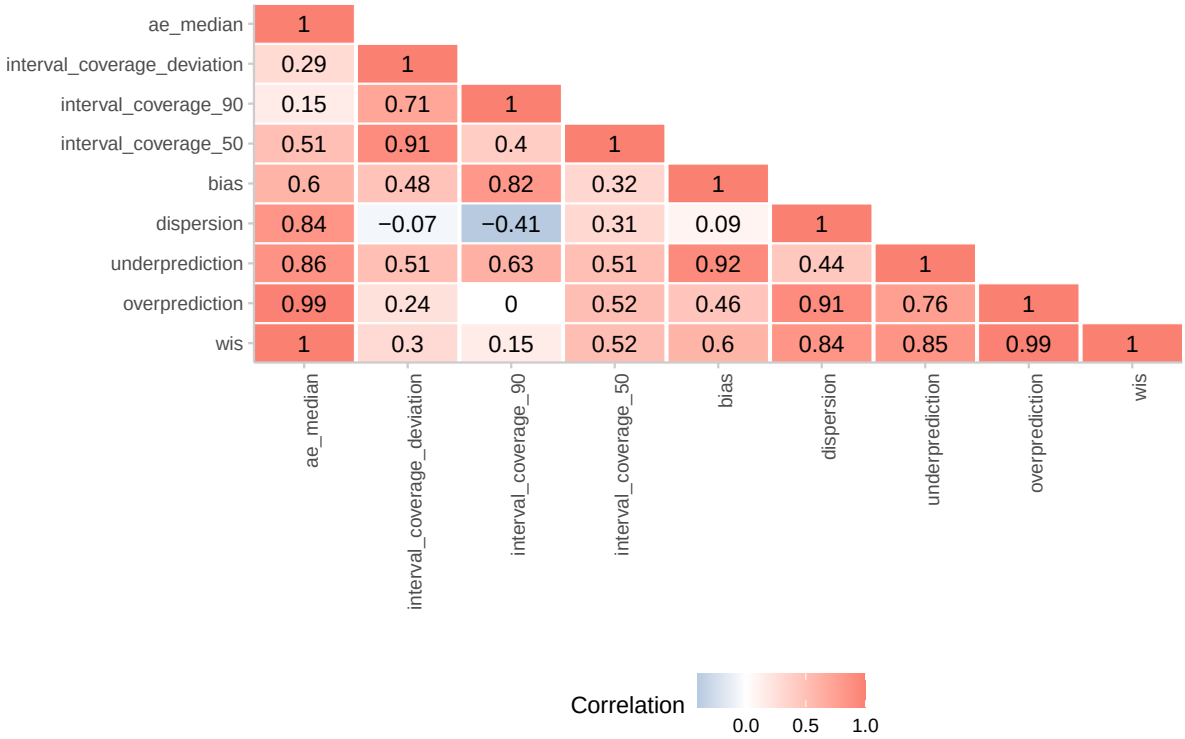

Figure 8: Plot of correlations between different scores. Numbers, as well as the shade of the cells, indicate the correlation between two scores.

a summarised score for each combination of the value in the specified columns (e.g. `by = c("model", "target_type")` will return one summarised score per model and target type). Equivalently, users can specify the columns that should be aggregated over (using the argument `across`). To display scores it is often useful to round the output, for example to two significant digits, which can be achieved with another call to `summarise_scores()`.

```
R> forecast_quantile |>
+   score(metrics = list("wis" = wis)) |>
+   summarise_scores(by = c("model", "target_type")) |>
+   summarise_scores(fun = signif, digits = 2)

                  model   wis
                 <char> <num>
1: EuroCOVIDhub-ensemble 17000
2: EuroCOVIDhub-ensemble    41
3: EuroCOVIDhub-baseline 29000
4: EuroCOVIDhub-baseline   160
5:  epiforecasts-EpiNow2 21000
6:  epiforecasts-EpiNow2    69
7:       UMass-MechBayes    52
```

While `summarise_scores()` accepts arbitrary summary functions, care has to be taken when using something else than `mean()`, as this may create an incentive for dishonest reporting.

Many scoring rules for probabilistic forecasts are 'strictly proper scoring rules' (Gneiting and Raftery 2007), meaning that they are constructed such that they cannot be cheated and always incentivise the forecaster to report her honest belief about the future. Let's assume that a forecaster's true belief about the future corresponds to a predictive distribution F. Then, if F was the true data-generating process, a scoring rule would be proper if it ensures that no other forecast distribution G would yield a better expected score. If the scoring rule ensures that under F no other possible predictive distribution can achieve the same expected score as F, then it is called strictly proper. From the forecaster's perspective, any deviation from her true belief F leads to a worsening of expected scores. When using summary functions other than the mean, however, scores may lose their propriety (the property of incentivising honest reporting) and become cheatable. For example, the median of several individual scores (individually based on a strictly proper scoring rule) is usually not proper. A forecaster judged by the median of several scores may be incentivised to misrepresent their true belief in a way that is not true for the mean score.

The user must exercise additional caution and should usually avoid aggregating scores across categories which differ much in the magnitude of the quantity to forecast, as (depending on the scoring rule used) forecast errors usually increase with the order of magnitude of the forecast target. In the given example, looking at one score per model (i.e., specifying `by = c("model")`) is problematic, as overall aggregate scores would be dominated by case forecasts, while performance on deaths would have little influence. Similarly, aggregating over different forecast horizons is often ill-advised as the mean will be dominated by further ahead forecast horizons. In the previous function calls, we therefore decided to only analyse forecasts with a forecast horizon of two weeks.

4.2. Additional functionality for summarised scores

Heatmaps

To detect systematic patterns it may be useful to visualise a single score across several dimensions. The function `plot_heatmap()` can be used to create a heatmap that achieves this. The following produces a heatmap of bias values across different locations and forecast targets (output shown in Figure 9).

```
R> example_continuous[horizon == 2] |>
+   as_forecast() |>
+   score() |>
+   summarise_scores(by = c("model", "location", "target_type")) |>
+   plot_heatmap(x = "location", metric = "bias") +
+     facet_wrap(~ target_type)
```

Weighted interval score decomposition

For quantile-based forecasts, the weighted interval score (WIS, Bracher *et al.* 2021, see Section C in the Appendix) is a commonly used strictly proper scoring rule for forecasts in a quantile-based format. The score is the sum of three components: overprediction, underprediction and dispersion (width of the forecast). These can be visualised using the function `plot_wis()`, as shown in Figure 10.

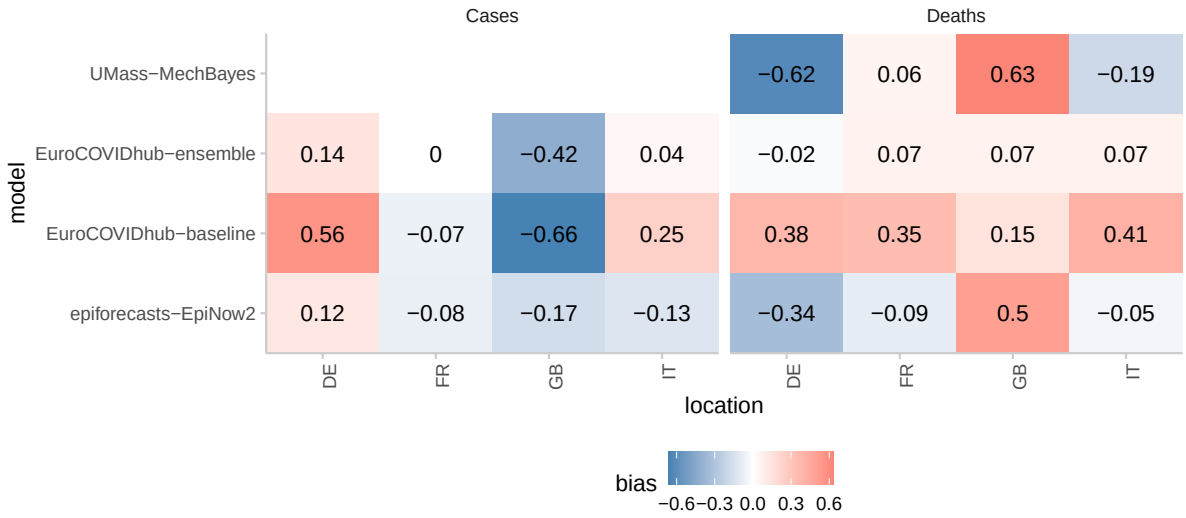

Figure 9: Heatmap of bias values for different models across different locations and forecast targets. Bias values are bound between -1 (underprediction) and 1 (overprediction) and should be 0 ideally. Red tiles indicate an upwards bias (overprediction), while blue tiles indicate a downwards bias (underprediction)

```
R> forecast_quantile |>
+   score() |>
+   summarise_scores(by = c("model", "target_type")) |>
+   plot_wis(relative_contributions = FALSE) +
+   facet_wrap(~ target_type,
+              scales = "free_x")
```

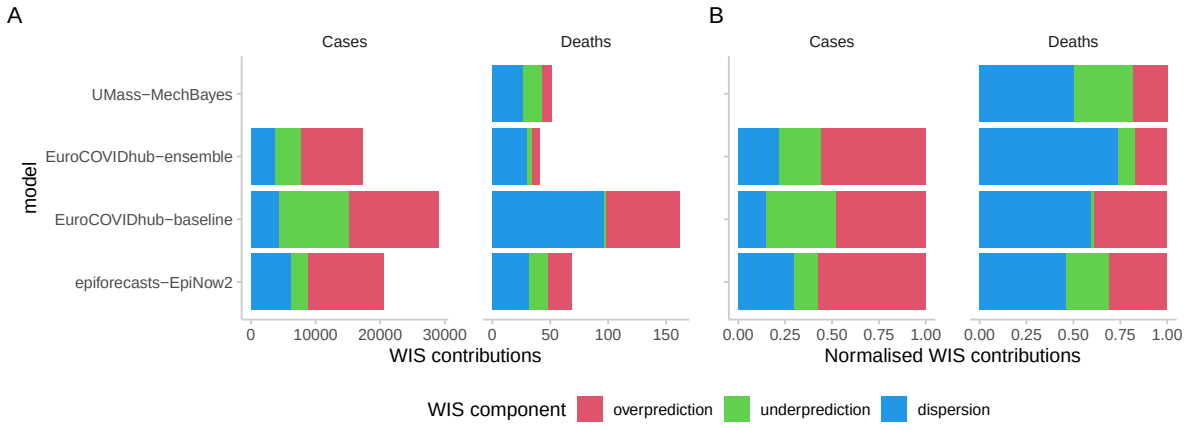

Figure 10: Decomposition of the weighted interval score (WIS) into dispersion, overprediction and underprediction. A: absolute contributions, B: contributions normalised to 1.

5. Discussion

Summary

This paper presented **scoringutils** an R package for forecast evaluation. It explained the core workflow, consisting of 1) validating and processing inputs, 2) scoring forecasts and 3) summarising scores, as well as additional functionality such as visualisation and diagnostic tools.

The package specialises in the evaluation of probabilistic forecasts (the forecast is a full predictive distribution). It provides a comprehensive framework based on **data.table** and allows users to validate, diagnose, visualise, transform and score forecasts using a wide range of default and custom scoring rules. The package is designed to be flexible and extensible, and to make it easy to use functionality from different packages in a single workflow. **scoringutils** addresses a gap in the existing ecosystem of forecast evaluation by creating a `data.table`-based forecast evaluation framework for probabilistic forecasts (similarly to what **yardstick** provides for point forecasts and classification tasks). Notably, **scoringutils** is the first package to provide extensive support for forecasts in a quantile-based forecasts, which is commonly used for example in Epidemiology. In addition to providing a coherent forecast evaluation workflow it offers a wide range of additional functions that practitioners may find useful when assessing or comparing the quality of their forecasts.

One important limitation of the package is that it currently does not support statistical testing of forecast performance as part of its core workflow. Determining whether a forecaster is significantly better than another is an important aspect of forecast evaluation that is currently mostly missing from the package. Another limitation is the fact that the package currently only supports a small set of possible types of forecasts. For example, forecasts in a bin-format or forecasts represented in a closed-form distribution (as can be scored for example using **scoringRules** are not supported. While it is in principle possible to extend the current classes and generic functions, this may not be very feasible in practice for most users. Some functionality in **scoringutils** is necessarily redundant with other packages that provide functionality to aid with the evaluation of forecasts. The overall idea of providing a `data.frame`-based evaluation framework, for example, is similar to what **yardstick** offers (albeit with a focus on point forecasts and classification tasks, rather than probabilistic forecasts). Having a single package that encompasses all possible use cases might be preferable. At the moment, **scoringutils** falls somewhat short of its aspiration to become a bridge between different packages in the forecast evaluation ecosystem. It does not yet offer a wide range of helper functions that allow users to easily convert between different formats and use functionality from other packages and many visualisations that are available in other packages, particularly with respect to model calibration, are missing.

A variety of extensions are planned for **scoringutils**. The first is the expansion of the forecast types that are supported. We plan to add support for evaluating categorical forecasts, as well as multivariate forecasts that specify a joint distribution across targets. Adding the possibility to score closed-form distributions might be another useful extension. A second area of expansion is the integration with other forecast evaluation and modelling packages. We aim to provide a variety of helper functions to convert to and from different formats, such as the one used by **yardstick** or formats used by modelling packages such as **odin**. These functions would make it easy to integrate **scoringutils** into existing workflows or use functionality from other packages that is not available in **scoringutils**. A third area of improvement is the addition of case studies and vignettes that make working with and extending functionality

from the package easier.

scoringutils is already used by a variety of public health institutions such as the US Centers for Disease Control, the European Centre for Disease Prevention and Control, as well as various academic institutions. The package is actively maintained and developed and we hope it will continue to be a valuable resource for researchers and practitioners working on forecast evaluation.

6. Acknowledgments

Funding statements

NIB received funding from the Health Protection Research Unit (grant code NIHR200908). HG MISSING. AC acknowledges funding by the NIHR, the Sergei Brin foundation, USAID, and the Academy of Medical Sciences. EvL acknowledges funding by the National Institute for Health Research (NIHR) Health Protection Research Unit (HPRU) in Modelling and Health Economics (grant number NIHR200908) and the European Union's Horizon 2020 research and innovation programme - project EpiPose (101003688). SF's work was supported by the Wellcome Trust (grant: 210758/Z/18/Z), and the NIHR (NIHR200908). SA's work was funded by the Wellcome Trust (grant: 210758/Z/18/Z). This study is partially funded by the National Institute for Health Research (NIHR) Health Protection Research Unit in Modelling and Health Economics, a partnership between UK Health Security Agency and Imperial College London in collaboration with LSHTM (grant code NIHR200908); and acknowledges funding from the MRC Centre for Global Infectious Disease Analysis (reference MR/R015600/1), jointly funded by the UK Medical Research Council (MRC) and the UK Foreign, Commonwealth & Development Office (FCDO), under the MRC/FCDO Concordat agreement and is also part of the EDCTP2 programme supported by the European Union. Disclaimer: "The views expressed are those of the author(s) and not necessarily those of the NIHR, UKHSA or the Department of Health and Social Care. We thank Community Jameel for Institute and research funding

References

Angus JE (1994). "The Probability Integral Transform and Related Results." *SIAM Review*, **36**(4), 652–654. ISSN 0036-1445. doi:10.1137/1036146.

Bosse NI, Abbott S, Cori A, van Leeuwen E, Bracher J, Funk S (2023). "Scoring Epidemiological Forecasts on Transformed Scales." *PLOS Computational Biology*, **19**(8), e1011393. ISSN 1553-7358. doi:10.1371/journal.pcbi.1011393.

Bracher J, Ray EL, Gneiting T, Reich NG (2021). "Evaluating Epidemic Forecasts in an Interval Format." *PLoS computational biology*, **17**(2), e1008618. ISSN 1553-7358. doi:10.1371/journal.pcbi.1008618.

Bracher J, Wolffram D, Deuschel J, Görgen K, Ketterer JL, Ullrich A, Abbott S, Barbarossa MV, Bertsimas D, Bhatia S, Bodych M, Bosse NI, Burgard JP, Castro L, Fairchild G, Fiedler J, Fuhrmann J, Funk S, Gambin A, Gogolewski K, Heyder S, Hotz T, Kheifetz

Y, Kirsten H, Krueger T, Krymova E, Leithäuser N, Li ML, Meinke JH, Miasojedow B, Michaud IJ, Mohring J, Nouvellet P, Nowosielski JM, Ozanski T, Radwan M, Rakowski F, Scholz M, Soni S, Srivastava A, Gneiting T, Schienle M (2022). "National and Subnational Short-Term Forecasting of COVID-19 in Germany and Poland during Early 2021." *Communications Medicine*, **2**(1), 1–17. ISSN 2730-664X. doi:10.1038/s43856-022-00191-8.

Cramer E, Ray EL, Lopez VK, Bracher J, Brennen A, Rivadeneira AJC, Gerding A, Gneiting T, House KH, Huang Y, Jayawardena D, Kanji AH, Khandelwal A, Le K, Mühlemann A, Niemi J, Shah A, Stark A, Wang Y, Wattanachit N, Zorn MW, Gu Y, Jain S, Bannur N, Deva A, Kulkarni M, Merugu S, Raval A, Shingi S, Tiwari A, White J, Woody S, Dahan M, Fox S, Gaither K, Lachmann M, Meyers LA, Scott JG, Tec M, Srivastava A, George GE, Cegan JC, Dettwiller ID, England WP, Farthing MW, Hunter RH, Lafferty B, Linkov I, Mayo ML, Parno MD, Rowland MA, Trump BD, Corsetti SM, Baer TM, Eisenberg MC, Falb K, Huang Y, Martin ET, McCauley E, Myers RL, Schwarz T, Sheldon D, Gibson GC, Yu R, Gao L, Ma Y, Wu D, Yan X, Jin X, Wang YX, Chen Y, Guo L, Zhao Y, Gu Q, Chen J, Wang L, Xu P, Zhang W, Zou D, Biegel H, Lega J, Snyder TL, Wilson DD, McConnell S, Walraven R, Shi Y, Ban X, Hong QJ, Kong S, Turtle JA, Ben-Nun M, Riley P, Riley S, Koyluoglu U, DesRoches D, Hamory B, Kyriakides C, Leis H, Milliken J, Moloney M, Morgan J, Ozcan G, Schrader C, Shakhnovich E, Siegel D, Spatz R, Stiefeling C, Wilkinson B, Wong A, Gao Z, Bian J, Cao W, Ferres JL, Li C, Liu TY, Xie X, Zhang S, Zheng S, Vespignani A, Chinazzi M, Davis JT, Mu K, y Piontti AP, Xiong X, Zheng A, Baek J, Farias V, Georgescu A, Levi R, Sinha D, Wilde J, Penna ND, Celi LA, Sundar S, Cavany S, España G, Moore S, Oidtman R, Perkins A, Osthus D, Castro L, Fairchild G, Michaud I, Karlen D, Lee EC, Dent J, Grantz KH, Kaminsky J, Kaminsky K, Keegan LT, Lauer SA, Lemaitre JC, Lessler J, Meredith HR, Perez-Saez J, Shah S, Smith CP, Truelove SA, Wills J, Kinsey M, Obrecht RF, Tallaksen K, Burant JC, Wang L, Gao L, Gu Z, Kim M, Li X, Wang G, Wang Y, Yu S, Reiner RC, Barber R, Gaikedu E, Hay S, Lim S, Murray C, Pigott D, Prakash BA, Adhikari B, Cui J, Rodríguez A, Tabassum A, Xie J, Keskinocak P, Asplund J, Baxter A, Oruc BE, Serban N, Arik SO, Dusenberry M, Epshteyn A, Kanal E, Le LT, Li CL, Pfister T, Sava D, Sinha R, Tsai T, Yoder N, Yoon J, Zhang L, Abbott S, Bosse NI, Funk S, Hellewel J, Meakin SR, Munday JD, Sherratt K, Zhou M, Kalantari R, Yamana TK, Pei S, Shaman J, Ayer T, Adee M, Chhatwal J, Dalgic OO, Ladd MA, Linas BP, Mueller P, Xiao J, Li ML, Bertsimas D, Lami OS, Soni S, Bouardi HT, Wang Y, Wang Q, Xie S, Zeng D, Green A, Bien J, Hu AJ, Jahja M, Narasimhan B, Rajanala S, Rumack A, Simon N, Tibshirani R, Tibshirani R, Ventura V, Wasserman L, O'Dea EB, Drake JM, Pagano R, Walker JW, Slayton RB, Johansson M, Biggerstaff M, Reich NG (2021). "Evaluation of Individual and Ensemble Probabilistic Forecasts of COVID-19 Mortality in the US." *medRxiv*, p. 2021.02.03.21250974. doi:10.1101/2021.02.03.21250974.

Cramer E, Reich NG, Wang SY, Niemi J, Hannan A, House K, Gu Y, Xie S, Horstman S, aniruddhadiga, Walraven R, starkari, Li ML, Gibson G, Castro L, Karlen D, Wattanachit N, jinghuichen, zyt9lsb, aagarwal1996, Woody S, Ray E, Xu FT, Biegel H, GuidoEspana, X X, Bracher J, Lee E, har96, leyouz (2020). "COVID-19 Forecast Hub: 4 December 2020 Snapshot." doi:10.5281/zenodo.3963371.

Czado C, Gneiting T, Held L (2009). "Predictive Model Assessment for Count Data." *Biometrics*, **65**(4), 1254–1261. ISSN 1541-0420. doi:10.1111/j.1541-0420.2009.01191.x.

Dawid AP (1984). "Present Position and Potential Developments: Some Personal Views Statistical Theory the Prequential Approach." *Journal of the Royal Statistical Society: Series A (General)*, **147**(2), 278–290. ISSN 2397-2327. doi:10.2307/2981683.

Dowle M, Srinivasan A (2023). *data.table: Extension of 'data.frame'*. R package version 1.14.8, URL https://CRAN.R-project.org/package=data.table.

Elliott G, Timmermann A (2016). "Forecasting in Economics and Finance." *Annual Review of Economics*, **8**(1), 81–110. doi:10.1146/annurev-economics-080315-015346.

Epstein ES (1969). "A Scoring System for Probability Forecasts of Ranked Categories." *Journal of Applied Meteorology*, **8**(6), 985–987. ISSN 0021-8952. doi:10.1175/1520-0450(1969)008<0985:ASSFPF>2.0.CO;2.

Funk S, Abbott S, Atkins BD, Baguelin M, Baillie JK, Birrell P, Blake J, Bosse NI, Burton J, Carruthers J, Davies NG, Angelis DD, Dyson L, Edmunds WJ, Eggo RM, Ferguson NM, Gaythorpe K, Gorsich E, Guyver-Fletcher G, Hellewell J, Hill EM, Holmes A, House TA, Jewell C, Jit M, Jombart T, Joshi I, Keeling MJ, Kendall E, Knock ES, Kucharski AJ, Lythgoe KA, Meakin SR, Munday JD, Openshaw PJM, Overton CE, Pagani F, Pearson J, Perez-Guzman PN, Pellis L, Scarabel F, Semple MG, Sherratt K, Tang M, Tildesley MJ, van Leeuwen E, Whittles LK, Group CCW, Team ICCR, Investigators I (2020). "Short-Term Forecasts to Inform the Response to the Covid-19 Epidemic in the UK." *medRxiv*, p. 2020.11.11.20220962. doi:10.1101/2020.11.11.20220962.

Funk S, Camacho A, Kucharski AJ, Lowe R, Eggo RM, Edmunds WJ (2019). "Assessing the Performance of Real-Time Epidemic Forecasts: A Case Study of Ebola in the Western Area Region of Sierra Leone, 2014-15." *PLOS Computational Biology*, **15**(2), e1006785. ISSN 1553-7358. doi:10.1371/journal.pcbi.1006785.

Gneiting T, Balabdaoui F, Raftery AE (2007). "Probabilistic Forecasts, Calibration and Sharpness." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **69**(2), 243–268. ISSN 1467-9868. doi:10.1111/j.1467-9868.2007.00587.x.

Gneiting T, Raftery AE (2005). "Weather Forecasting with Ensemble Methods." *Science*, **310**(5746), 248–249. ISSN 0036-8075, 1095-9203. doi:10.1126/science.1115255.

Gneiting T, Raftery AE (2007). "Strictly Proper Scoring Rules, Prediction, and Estimation." *Journal of the American Statistical Association*, **102**(477), 359–378. ISSN 0162-1459, 1537-274X. doi:10.1198/016214506000001437.

Good IJ (1952). "Rational Decisions." *Journal of the Royal Statistical Society. Series B (Methodological)*, **14**(1), 107–114. ISSN 0035-9246. 2984087.

Hamill TM (2001). "Interpretation of Rank Histograms for Verifying Ensemble Forecasts." *Monthly Weather Review*, **129**(3), 550–560. ISSN 1520-0493, 0027-0644. doi:10.1175/1520-0493(2001)129<0550:IORHFV>2.0.CO;2.

Hamner B, Frasco M (2018). *Metrics: Evaluation Metrics for Machine Learning*. R package version 0.1.4, URL https://CRAN.R-project.org/package=Metrics.

Jordan A, Krüger F, Lerch S (2019). "Evaluating Probabilistic Forecasts with scoringRules." *Journal of Statistical Software*, **90**(12), 1–37. doi:10.18637/jss.v090.i12.

Kuhn M, Vaughan D, Hvitfeldt E (2023a). *yardstick: Tidy Characterizations of Model Performance*. R package version 1.2.0, URL https://CRAN.R-project.org/package=yardstick.

Kuhn M, Vaughan D, Ruiz E (2023b). *probably: Tools for Post-Processing Class Probability Estimates*. R package version 1.0.2, URL https://CRAN.R-project.org/package=probably.

Kuhn M, Wickham H (2020). *Tidymodels: a collection of packages for modeling and machine learning using tidyverse principles*. URL https://www.tidymodels.org.

Kukkonen J, Olsson T, Schultz DM, Baklanov A, Klein T, Miranda AI, Monteiro A, Hirtl M, Tarvainen V, Boy M, Peuch VH, Poupkou A, Kioutsioukis I, Finardi S, Sofiev M, Sokhi R, Lehtinen KEJ, Karatzas K, San José R, Astitha M, Kallos G, Schaap M, Reimer E, Jakobs H, Eben K (2012). "A Review of Operational, Regional-Scale, Chemical Weather Forecasting Models in Europe." *Atmospheric Chemistry and Physics*, **12**(1), 1–87. ISSN 1680-7316. doi:10.5194/acp-12-1-2012.

Laboratory NRA (2015). *verification: Weather Forecast Verification Utilities*. R package version 1.42, URL https://CRAN.R-project.org/package=verification.

Liboschik T, Fokianos K, Fried R (2017). "tscount: An R Package for Analysis of Count Time Series Following Generalized Linear Models." *Journal of Statistical Software*, **82**(5), 1–51. doi:10.18637/jss.v082.i05.

Mann HB, Whitney DR (1947). "On a Test of Whether One of Two Random Variables Is Stochastically Larger than the Other." *The Annals of Mathematical Statistics*, **18**(1), 50–60. ISSN 0003-4851, 2168-8990. doi:10.1214/aoms/1177730491.

Matheson JE, Winkler RL (1976). "Scoring Rules for Continuous Probability Distributions." *Management Science*, **22**(10), 1087–1096. ISSN 0025-1909. doi:10.1287/mnsc.22.10.1087.

Merkle EC, Steyvers M (2013). "Choosing a Strictly Proper Scoring Rule." *Decision Analysis*, **10**, 292–304.

Meyer S, Held L, Höhle M (2017). "Spatio-Temporal Analysis of Epidemic Phenomena Using the R Package surveillance." *Journal of Statistical Software*, **77**(11), 1–55. doi:10.18637/jss.v077.i11.

Murphy AH (1971). "A Note on the Ranked Probability Score." *Journal of Applied Meteorology and Climatology*, **10**(1), 155–156. ISSN 1520-0450. doi:10.1175/1520-0450(1971)010<0155:ANOTRP>2.0.CO;2.

O'Hara-Wild M, Hyndman R, Wang E (2023). *fabletools: Core Tools for Packages in the 'fable' Framework*. R package version 0.3.4, URL https://CRAN.R-project.org/package=fabletools.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Reich NG, Brooks LC, Fox SJ, Kandula S, McGowan CJ, Moore E, Osthus D, Ray EL, Tushar A, Yamana TK, Biggerstaff M, Johansson MA, Rosenfeld R, Shaman J (2019). "A Collaborative Multiyear, Multimodel Assessment of Seasonal Influenza Forecasting in the United States." *Proceedings of the National Academy of Sciences*, **116**(8), 3146–3154. ISSN 0027-8424, 1091-6490. doi:10.1073/pnas.1812594116.

Rizopoulos D (2019). *cvGEE: Cross-Validated Predictions from GEE*. R package version 0.3-0, URL https://CRAN.R-project.org/package=cvGEE.

Rizopoulos D (2023). *GLMMadaptive: Generalized Linear Mixed Models using Adaptive Gaussian Quadrature*. R package version 0.9-0, URL https://CRAN.R-project.org/package=GLMMadaptive.

Sadatsafavi M, Safari A, Lee TY (2023). *predtools: Prediction Model Tools*. R package version 0.0.3, URL https://CRAN.R-project.org/package=predtools.

Sherratt K, Gruson H, Grah R, Johnson H, Niehus R, Prasse B, Sandman F, Deuschel J, Wolffram D, Abbott S, Ullrich A, Gibson G, Ray EL, Reich NG, Sheldon D, Wang Y, Wattanachit N, Wang L, Trnka J, Obozinski G, Sun T, Thanou D, Pottier L, Krymova E, Barbarossa MV, Leithäuser N, Mohring J, Schneider J, Wlazlo J, Fuhrmann J, Lange B, Rodiah I, Baccam P, Gurung H, Stage S, Suchoski B, Budzinski J, Walraven R, Villanueva I, Tucek V, Šmíd M, Zajícek M, Pérez ÁC, Reina B, Bosse NI, Meakin S, Di Loro A, Maruotti A, Eclerová V, Kraus A, Kraus D, Pribylova L, Dimitris B, Li ML, Saksham S, Dehning J, Mohr S, Priesemann V, Redlarski G, Bejar B, Ardenghi G, Parolini N, Ziarelli G, Bock W, Heyder S, Hotz T, E SD, Guzman-Merino M, Aznarte JL, Moriña D, Alonso S, Álvarez E, López D, Prats C, Burgard JP, Rodloff A, Zimmermann T, Kuhlmann A, Zibert J, Pennoni F, Divino F, Català M, Lovison G, Giudici P, Tarantino B, Bartolucci F, Jona LG, Mingione M, Farcomeni A, Srivastava A, Montero-Manso P, Adiga A, Hurt B, Lewis B, Marathe M, Porebski P, Venkatramanan S, Bartczuk R, Dreger F, Gambin A, Gogolewski K, Gruziel-Slomka M, Krupa B, Moszynski A, Niedzielewski K, Nowosielski J, Radwan M, Rakowski F, Semeniuk M, Szczurek E, Zielinski J, Kisielewski J, Pabjan B, Holger K, Kheifetz Y, Scholz M, Bodych M, Filinski M, Idzikowski R, Krueger T, Ozanski T, Bracher J, Funk S (2022). "Predictive Performance of Multi-Model Ensemble Forecasts of COVID-19 across European Nation." *Europe PMC*. doi:10.1101/2022.06.16.22276024.

Siegert S (2020). *SpecsVerification: Forecast Verification Routines for Ensemble Forecasts of Weather and Climate*. R package version 0.5-3, URL https://CRAN.R-project.org/package=SpecsVerification.

Timmermann A (2018). "Forecasting Methods in Finance." *Annual Review of Financial Economics*, **10**(1), 449–479. doi:10.1146/annurev-financial-110217-022713.

Yan Y (2016). *MLmetrics: Machine Learning Evaluation Metrics*. R package version 1.1.1, URL https://CRAN.R-project.org/package=MLmetrics.

Zeileis A, Lang MN (2022). *topmodels: Infrastructure for Inference and Forecasting in Probabilistic Models*. R package version 0.1-0/r1498, URL https://R-Forge.R-project.org/projects/topmodels/.

A. Constructing and validating forecast objects

The following section gives an overview of how **scoringutils** constructs forecast objects. The `forecast` class comes with a constructor, `new_forecast()`, a generic validation function, `validate_forecast()`, and a convenient wrapper function `as_forecast()`.

`new_forecast()` constructs a `forecast` object based on a `data.frame` or similar. It makes a deep copy of the inptut and converts it into a `data.table`, adds a `model` column with value "Unspecified model" if there isn't one and adds a class `forecast_*`, where `*` depends on the forecast type to the object.

`validate_forecast()` is a generic which dispatches to a specialised validator method depending on the class of the input. It validates the input and returns it if it is valid. If the input is not valid, it throws an error with a message that explains what went wrong.

`as_forecast()` (optionally) renames existing columns to conform with the requirements for forecast objects, (optionally) sets the forecast unit, determines the forecast type of the input (and optionally checks for consistency with what the user expects), constructs the class and validates the input. The process is illustrated in Figure A.11.

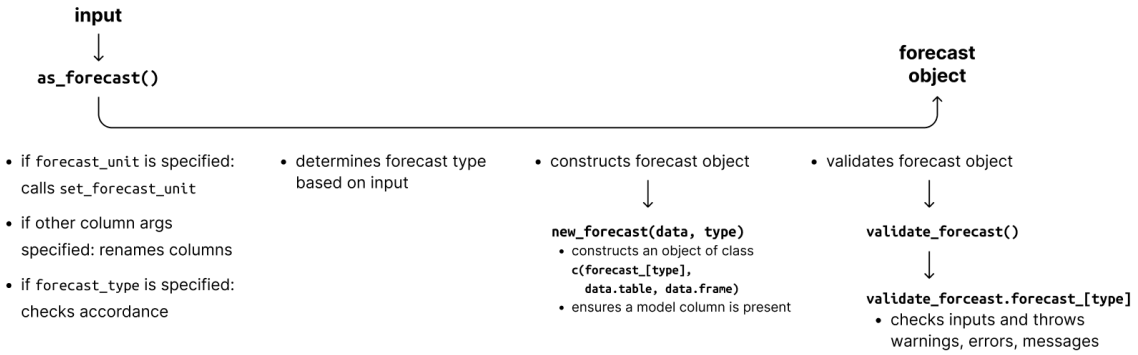

Figure A.11: Illustration of the process of creating a 'forecast' object.

B. Comparing different calibration plots

The following Figure gives a more detailed overview of how to interpret different calibration plots (showing the actual forecasts and observations that produced the corresponding visualisations).

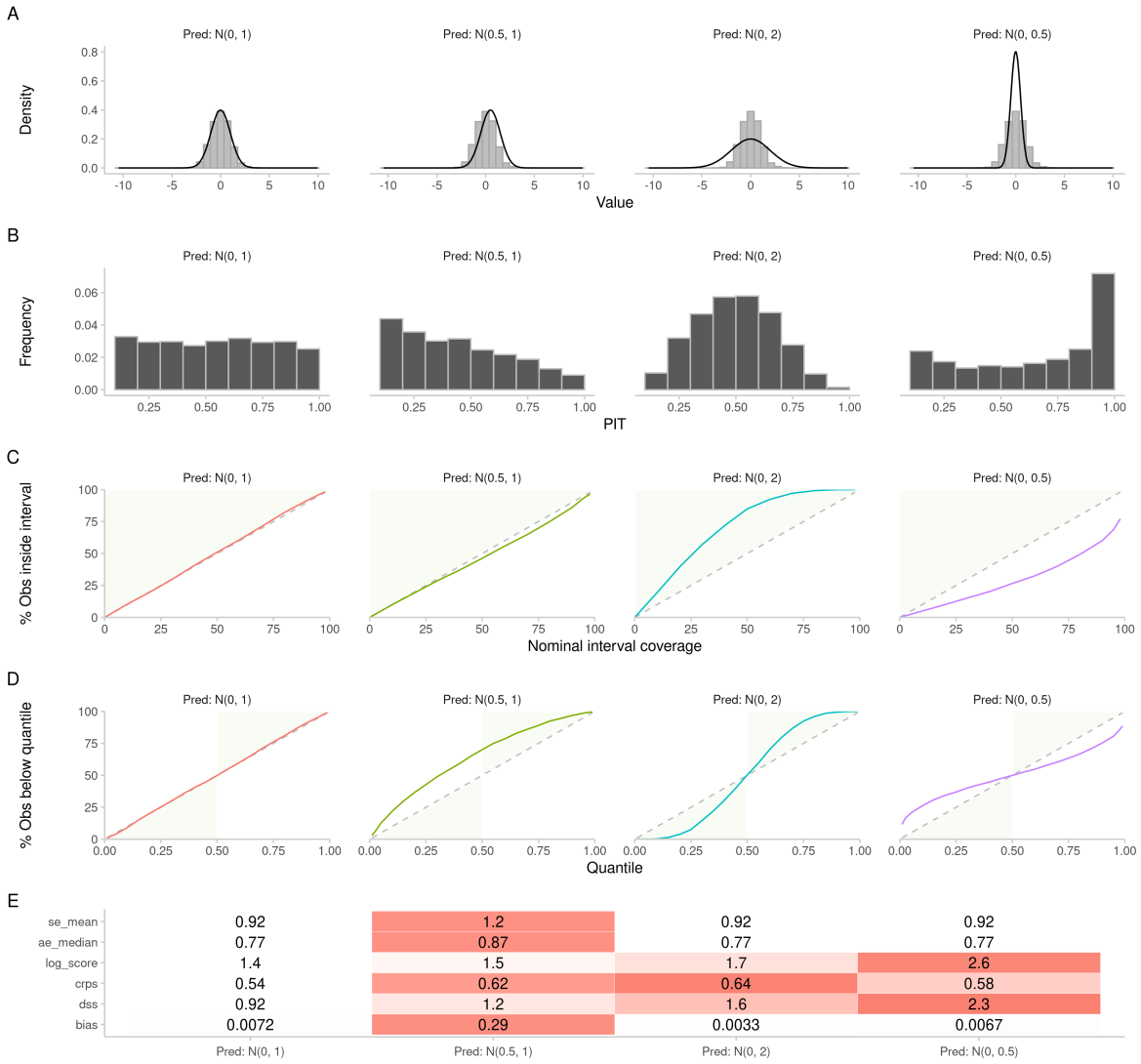

Figure B.12: A: Different forecasting distributions (black) against observations sampled from a standard normal distribution (grey histograms). B: PIT histograms based on the predictive distributions and the sampled observations shown in A. C: Empirical vs. nominal coverage of the central prediction intervals for simulated observations and predictions. Areas shaded in green indicate that the forecasts are too wide (i.e., underconfident), covering more true values than they actually should, while areas in white indicate that the model generates too narrow predictions and fails to cover the desired proportion of true values with its prediction intervals. D: Quantile coverage values, with green areas indicating too wide (i.e., conservative) forecasts. E: Scores for the standard normal predictive distribution and the observations drawn from different data-generating distributions.

C. Details on the weighted interval score (WIS)

The WIS treats the predictive quantiles as a set of symmetric prediction intervals and measures the distance between the observation and the forecast interval. It can be decomposed into a dispersion (uncertainty) component and penalties for over- and underprediction. For a single interval, the interval score is computed as

$$IS_\alpha(F, y) = \underbrace{(u - l)}_{\text{dispersion}} + \underbrace{\frac{2}{\alpha} \cdot (l - y) \cdot \mathbf{1}(y \leq l)}_{\text{overprediction}} + \underbrace{\frac{2}{\alpha} \cdot (y - u) \cdot \mathbf{1}(y \geq u)}_{\text{underprediction}},$$

where $\mathbf{1}()$ is the indicator function, y is the observed value, and l and u are the $\frac{\alpha}{2}$ and $1 - \frac{\alpha}{2}$ quantiles of the predictive distribution F, i.e. the lower and upper bound of a single prediction interval. For a set of K prediction intervals and the median m, the score is computed as a weighted sum,

$$WIS = \frac{1}{K + 0.5} \cdot \left(w_0 \cdot |y - m| + \sum_{k=1}^{K} w_k \cdot IS_\alpha(F, y) \right),$$

where w_k is a weight for every interval. Usually, $w_k = \frac{\alpha_k}{2}$ and $w_0 = 0.5$.

*Nikos I. Bosse, Hugo Gruson, Anne Cori, Edwin van Leeuwen, Sebastian Funk, Sam Abbott*33

Affiliation:

Nikos I. Bosse
London School of Hygiene & Tropical Medicine (LSHTM)
Centre for Mathematical Modelling of Infectious Diseases
London School of Hygiene & Tropical Medicine
Keppel Street
London WC1E 7HT
E-mail: nikos.bosse@lshtm.ac.uk
URL: https://lshtm.ac.uk

Hugo Gruson
LSHTM
Centre for Mathematical Modelling of Infectious Diseases
London School of Hygiene & Tropical Medicine
Keppel Street
London WC1E 7HT
E-mail: hugo.gruson@lshtm.ac.uk

Anne Cori
Imperial College London
MRC Centre for Global Infectious Disease Analysis, School of Public Health
Imperial College London
Norfolk Place
London W2 1PG
E-mail: a.cori@imperial.ac.uk

Edwin van Leeuwen
UK Health Security Agency, LSHTM
Statistics, Modelling and Economics Department
UK Health Security Agency
London NW9 5EQ
E-mail: Edwin.VanLeeuwen@phe.gov.uk

Sebastian Funk
LSHTM
Centre for Mathematical Modelling of Infectious Diseases
London School of Hygiene & Tropical Medicine
Keppel Street
London WC1E 7HT
E-mail: sebastian.funk@lshtm.ac.uk

Sam Abbott
LSHTM
Centre for Mathematical Modelling of Infectious Diseases
London School of Hygiene & Tropical Medicine
Keppel Street
London WC1E 7HT
E-mail: sam.abbott@lshtm.ac.uk