# The `hubEnsembles` package

# Introduction

The `hubEnsembles` package includes functionality for aggregating model outputs, such as forecasts or projections, that are submitted to a hub by multiple models and combined into ensemble model outputs. The package includes two main functions: `simple_ensemble` and `linear_pool`. We illustrate these functions in this vignette, and briefly compare them.

This vignette uses the following R packages:

```
library(dplyr)
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
library(plotly)
#> Loading required package: ggplot2
#>
#> Attaching package: 'plotly'
#> The following object is masked from 'package:ggplot2':
#>
#>     last_plot
#> The following object is masked from 'package:stats':
#>
#>     filter
#> The following object is masked from 'package:graphics':
#>
#>     layout
library(hubUtils)
library(hubEnsembles)
```

# Example data: a simple forecast hub

The `example-simple-forecast-hub` has been created by the Consortium of Infectious Disease Modeling Hubs as a simple example hub to demonstrate the set up and functionality for the hubverse. The hub includes both target data and example model output data.

```
hub_path <- system.file("example-data/example-simple-forecast-hub",
                        package = "hubEnsembles")

model_outputs <- hubUtils::connect_hub(hub_path) %>%
  dplyr::collect()
head(model_outputs)
#> # A tibble: 6 × 8
#>   origin_date horizon location target  output_type output_type_id value model_id
#>   <date>        <int> <chr>    <chr>   <chr>               <dbl> <int> <chr>
#> 1 2022-12-05       -6 20       inc co… quantile             0.01     22 UMass-ar
#> 2 2022-12-05       -6 20       inc co… quantile             0.025    24 UMass-ar
#> 3 2022-12-05       -6 20       inc co… quantile             0.05     26 UMass-ar
#> 4 2022-12-05       -6 20       inc co… quantile             0.1      28 UMass-ar
#> 5 2022-12-05       -6 20       inc co… quantile             0.15     30 UMass-ar
#> 6 2022-12-05       -6 20       inc co… quantile             0.2      32 UMass-ar

target_data_path <- file.path(hub_path, "target-data",
                              "covid-hospitalizations.csv")
target_data <- read.csv(target_data_path)
head(target_data)
#>     time_idx location value         target
#> 1 2021-03-21       46    12 inc covid hosp
#> 2 2021-03-04       45    82 inc covid hosp
#> 3 2021-02-26       46     7 inc covid hosp
#> 4 2021-02-20       44    21 inc covid hosp
#> 5 2021-02-09       44    19 inc covid hosp
#> 6 2021-01-25       25   224 inc covid hosp
```

# Creating ensembles with `simple_ensemble`

The `simple_ensemble` function is used to summarize across component model outputs; this function can be applied to predictions with an `output_type` of `mean`, `median`, `quantile`, `cdf`, or `pmf`.

The `simple_ensemble` function defaults to calculating an equally weighted mean across all component model outputs for each unique `output_type_id`. For our example data, which contains two output types (`median` and `quantile`), this means the resulting ensemble will be the mean of component model submitted values for each quantile.

```
mean_ens <- hubEnsembles::simple_ensemble(model_outputs)
head(mean_ens)
#> # A tibble: 6 × 8
#>   model_id  origin_date horizon location target output_type output_type_id value
#>   <chr>     <date>        <int> <chr>    <chr>  <chr>                <dbl> <dbl>
#> 1 hub-ense… 2022-12-05       -6 20       inc c… median              NA      37.3
#> 2 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.01   14.7
#> 3 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.025  15.7
#> 4 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.05   17
#> 5 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.1    18.3
#> 6 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.15   21.7
```

## Changing the aggregation function

We can change the function used to aggregate across model outputs. For example, we may want to calculate a median of component model submitted values for each quantile. We will also use the `model_id` argument to distinguish this ensemble.

```
median_ens <- hubEnsembles::simple_ensemble(model_outputs,
                                            agg_fun = median,
                                            model_id = "hub-ensemble-median")
head(median_ens)
#> # A tibble: 6 × 8
#>   model_id  origin_date horizon location target output_type output_type_id value
#>   <chr>     <date>        <int> <chr>    <chr>  <chr>                <dbl> <int>
#> 1 hub-ense… 2022-12-05       -6 20       inc c… median              NA        37
#> 2 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.01     22
#> 3 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.025    23
#> 4 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.05     25
#> 5 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.1      27
#> 6 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.15     28
```

Custom functions can also be passed into the `agg_fun` argument. For example, a geometric mean may be a more appropriate way to combine component model outputs. Any custom function to be used requires an argument `x` for the vector of numeric values to summarize, and if relevant, an argument `w` of numeric weights.

```
geometric_mean <- function(x){
    n <- length(x)
    return(prod(x)^(1/n))
}

geometric_mean_ens <-  hubEnsembles::simple_ensemble(model_outputs,
                                            agg_fun = geometric_mean,
                                            model_id = "hub-ensemble-geometric")
head(geometric_mean_ens)
#> # A tibble: 6 × 8
#>   model_id  origin_date horizon location target output_type output_type_id value
#>   <chr>     <date>        <int> <chr>    <chr>  <chr>                <dbl> <dbl>
#> 1 hub-ense… 2022-12-05       -6 20       inc c… median              NA      37.3
#> 2 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.01    0
#> 3 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.025   0
#> 4 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.05    0
#> 5 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.1     0
#> 6 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.15   18.0
```

## Weighting model contributions

In addition, we can weight the contributions of each model by providing a table of weights, which are provided in a `data.frame` with a `model_id` column and a `weight` column.

```
model_weights <- data.frame(model_id = c("UMass-ar", "UMass-gbq", "simple_hub-baseline"),
                            weight = c(0.4, 0.4, 0.2))

weighted_mean_ens <- hubEnsembles::simple_ensemble(model_outputs,
                                                   weights = model_weights,
```

```
                                                  model_id = "hub-ensemble-weighted-mean")
head(weighted_mean_ens)
#> # A tibble: 6 × 8
#>   model_id  origin_date horizon location target output_type output_type_id value
#>   <chr>     <date>        <int> <chr>    <chr>  <chr>                <dbl> <dbl>
#> 1 hub-ense… 2022-12-05       -6 20       inc c… median              NA      37.6
#> 2 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.01   17.6
#> 3 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.025  18.8
#> 4 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.05   20.4
#> 5 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.1    22
#> 6 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.15   24.6
```

# Creating ensembles with `linear_pool`

An alternative approach to generate an ensemble is a linear pool, or a distributional mixture; this function can be applied to predictions with an `output_type` of `mean`, `quantile`, `cdf`, or `pmf`. Our example hub includes `median` output type, so we exclude it from the calculation.

For `mean`, `cdf` and `pmf` output types, the linear pool is equivalent to using a mean `simple_ensemble`. For `quantile` model outputs, the `linear_pool` function needs to approximate a full probability distribution using the value-quantile pairs from each component model. As a default, this is done with functions in the `distfromq` package, which defaults to fitting a monotonic cubic spline.

```
linear_pool_ens <- hubEnsembles::linear_pool(model_outputs %>%
                                               filter(output_type != "median"))
head(linear_pool_ens)
#> # A tibble: 6 × 8
#>   model_id  origin_date horizon location target output_type output_type_id value
#>   <chr>     <date>        <int> <chr>    <chr>  <chr>                <dbl> <dbl>
#> 1 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.01   0
#> 2 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.025  0
#> 3 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.05   7.01
#> 4 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.1    21.1
#> 5 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.15   25.6
#> 6 hub-ense… 2022-12-05       -6 20       inc c… quantile             0.2    27.6
```

# Plots

```
basic_plot_function <- function(plot_df, truth_df, plain_line = 0.5, ribbon = c(0.975, 0.025),
                                forecast_date) {

  plain_df <- dplyr::filter(plot_df, output_type_id == plain_line)

  ribbon_df <- dplyr::filter(plot_df, output_type_id %in% ribbon) %>%
    dplyr::mutate(output_type_id = ifelse(output_type_id == min(ribbon),
                                          "min", "max")) %>%
    tidyr::pivot_wider(names_from = output_type_id, values_from = value)

  plot_model <- plot_ly(height = 600, colors = scales::hue_pal()(50))

  if (!is.null(truth_df)) {
    plot_model <- plot_model %>%
      add_trace(data = truth_df, x = ~time_idx, y = ~value, type = "scatter",
                mode = "lines+markers", line = list(color = "#6e6e6e"),
                hoverinfo = "text", name = "ground truth",
                hovertext = paste("Date: ", truth_df$time_value, "<br>",
                                  "Ground truth: ",
                                  format(truth_df$value, big.mark = ","),
                                sep = ""),
                marker = list(color = "#6e6e6e", size = 7))
  }
  plot_model <- plot_model %>%
    add_lines(data = plain_df, x = ~target_date, y = ~value,
              color = ~model_id) %>%
    add_ribbons(data = ribbon_df, x = ~target_date, ymin = ~min,
                ymax = ~max, color = ~model_id, opacity = 0.25,
                line = list(width = 0), showlegend = FALSE) %>%
    plotly::layout(shapes = list(type = "line", y0 = 0, y1 = 1, yref = "paper",
                                 x0 = forecast_date, x1 = forecast_date,
                                 line = list(color = "gray")))
}


plot_df <- dplyr::bind_rows(model_outputs, mean_ens) %>%
  dplyr::filter(location == "US", origin_date == "2022-12-12") %>%
```

```
  dplyr::mutate(target_date = origin_date + horizon)

plot <- basic_plot_function(
    plot_df,
    truth_df = target_data %>%
        dplyr::filter(location == "US",
                      time_idx >= "2022-10-01",
                      time_idx <= "2023-03-01"),
    forecast_date = "2022-12-12")
plot
#> Warning: Can't display both discrete & non-discrete data on same axis
```