

# Package: modeltime.ensemble (via r-universe)

October 18, 2024

**Type** Package

**Title** Ensemble Algorithms for Time Series Forecasting with Modeltime

**Version** 1.0.4

**Description** A 'modeltime' extension that implements time series ensemble forecasting methods including model averaging, weighted averaging, and stacking. These techniques are popular methods to improve forecast accuracy and stability.

**URL** <https://business-science.github.io/modeltime.ensemble/>,  
<https://github.com/business-science/modeltime.ensemble>

**BugReports** <https://github.com/business-science/modeltime.ensemble/issues>

**License** MIT + file LICENSE

**Encoding** UTF-8

**Depends** modeltime (>= 1.2.3), modeltime.resample (>= 0.2.1), R (>= 3.5)

**Imports** tune (>= 0.1.2), rsample, yardstick, workflows (>= 0.2.1), recipes (>= 0.1.15), timetk (>= 2.5.0), tibble, dplyr (>= 1.0.0), tidyr, purrr, stringr, rlang (>= 0.1.2), cli, generics, magrittr, tictoc, parallel, doParallel, foreach, glmnet

**Suggests** gt, dials, utils, earth, testthat, tidymodels, xgboost, lubridate, knitr, rmarkdown

**RoxygenNote** 7.3.1

**VignetteBuilder** knitr

**Roxygen** list(markdown = TRUE)

**Repository** <https://business-science.r-universe.dev>

**RemoteUrl** <https://github.com/business-science/modeltime.ensemble>

**RemoteRef** HEAD

**RemoteSha** 37f80be857188c428b651453b99c0d4afd8b6273

## Contents

ensemble_average . . . . .	2
ensemble_model_spec . . . . .	3
ensemble_nested_average . . . . .	6
ensemble_nested_weighted . . . . .	8
ensemble_weighted . . . . .	10

<b>Index</b>	<b>12</b>
--------------	-----------

---

ensemble_average	<i>Creates an Ensemble Model using Mean/Median Averaging</i>
------------------	--

---

### Description

Creates an Ensemble Model using Mean/Median Averaging

### Usage

```
ensemble_average(object, type = c("mean", "median"))
```

### Arguments

object	A Modeltime Table
type	Specify the type of average ("mean" or "median")

### Details

The input to an ensemble\_average() model is always a Modeltime Table, which contains the models that you will ensemble.

#### Averaging Methods

The average method uses an un-weighted average using type of either:

- "mean": Performs averaging using mean(x, na.rm = TRUE) to aggregate each underlying models forecast at each timestamp
- "median": Performs averaging using stats::median(x, na.rm = TRUE) to aggregate each underlying models forecast at each timestamp

### Value

A mdl\_time\_ensemble object.

## Examples

```
library(tidymodels)
library(modeltime)
library(modeltime.ensemble)
library(dplyr)
library(timetk)

# Make an ensemble from a Modeltime Table
ensemble_fit <- m750_models %>%
  ensemble_average(type = "mean")

ensemble_fit

# Forecast with the Ensemble
modeltime_table(
  ensemble_fit
) %>%
  modeltime_forecast(
    new_data = testing(m750_splits),
    actual_data = m750
  ) %>%
  plot_modeltime_forecast(
    .interactive = FALSE,
    .conf_interval_show = FALSE
  )
```

---

ensemble\_model\_spec *Creates a Stacked Ensemble Model from a Model Spec*

---

## Description

A 2-stage stacking regressor that follows:

1. Stage 1: Sub-Model's are Trained & Predicted using `modeltime.resample::modeltime_fit_resamples()`.
2. Stage 2: A Meta-learner (`model_spec`) is trained on Out-of-Sample Sub-Model Predictions using `ensemble_model_spec()`.

## Usage

```
ensemble_model_spec(
  object,
  model_spec,
  kfolds = 5,
  param_info = NULL,
  grid = 6,
  control = control_grid()
)
```

**Arguments**

object	A Modeltime Table. Used for ensemble sub-models.
model_spec	A model_spec object defining the meta-learner stacking model specification to be used. Can be either: <ol style="list-style-type: none"> <li>1. <b>A non-tunable</b> model_spec: Parameters are specified and are not optimized via tuning.</li> <li>2. <b>A tunable</b> model_spec: Contains parameters identified for tuning with <code>tune::tune()</code></li> </ol>
kfolds	K-Fold Cross Validation for tuning the Meta-Learner. Controls the number of folds used in the meta-learner's cross-validation. Gets passed to <code>rsample::vfold_cv()</code> .
param_info	A <code>dials::parameters()</code> object or NULL. If none is given, a parameters set is derived from other arguments. Passing this argument can be useful when parameter ranges need to be customized.
grid	Grid specification or grid size for tuning the Meta Learner. Gets passed to <code>tune::tune_grid()</code> .
control	An object used to modify the tuning process. Uses <code>tune::control_grid()</code> by default. Use <code>control_grid(verbose = TRUE)</code> to follow the training process.

**Details****Stacked Ensemble Process**

- Start with a *Modeltime Table* to define your sub-models.
- Step 1: Use `modeltime.resample::modeltime_fit_resamples()` to perform the submodel resampling procedure.
- Step 2: Use `ensemble_model_spec()` to define and train the meta-learner.

**What goes on inside the Meta Learner?**

The Meta-Learner Ensembling Process uses the following basic steps:

1. **Make Cross-Validation Predictions.** Cross validation predictions are made for each sub-model with `modeltime.resample::modeltime_fit_resamples()`. The out-of-sample sub-model predictions contained in `.resample_results` are used as the input to the meta-learner.
2. **Train a Stacked Regressor (Meta-Learner).** The sub-model out-of-sample cross validation predictions are then modeled using a `model_spec` with options:
  - **Tuning:** If the `model_spec` does include tuning parameters via `tune::tune()` then the meta-learner will be hyperparameter tuned using K-Fold Cross Validation. The parameters and grid can adjusted using `kfolds`, `grid`, and `param_info`.
  - **No-Tuning:** If the `model_spec` does *not* include tuning parameters via `tune::tune()` then the meta-learner will not be hyperparameter tuned and will have the model fitted to the sub-model predictions.
3. **Final Model Selection.**
  - **If tuned,** the final model is selected based on RMSE, then retrained on the full set of out of sample predictions.

- **If not-tuned**, the fitted model from Stage 2 is used.

### Progress

The best way to follow the training process and watch progress is to use `control = control_grid(verbose = TRUE)` to see progress.

### Parallelize

Portions of the process can be parallelized. To parallelize, set up parallelization using `tune` via one of the backends such as `doFuture`. Then set `control = control_grid(allow_par = TRUE)`

### Value

A `mdl_time_ensemble` object.

### Examples

```
library(tidymodels)
library(modeltime)
library(modeltime.ensemble)
library(dplyr)
library(timetk)
library(glmnet)

# Step 1: Make resample predictions for submodels
resamples_tscv <- training(m750_splits) %>%
  time_series_cv(
    assess = "2 years",
    initial = "5 years",
    skip = "2 years",
    slice_limit = 1
  )

submodel_predictions <- m750_models %>%
  modeltime_fit_resamples(
    resamples = resamples_tscv,
    control = control_resamples(verbose = TRUE)
  )

# Step 2: Metalearner ----

# * No Metalearner Tuning
ensemble_fit_lm <- submodel_predictions %>%
  ensemble_model_spec(
    model_spec = linear_reg() %>% set_engine("lm"),
    control = control_grid(verbose = TRUE)
  )

ensemble_fit_lm

# * With Metalearner Tuning ----
ensemble_fit_glmnet <- submodel_predictions %>%
  ensemble_model_spec(
```

```

      model_spec = linear_reg(
        penalty = tune(),
        mixture = tune()
      ) %>%
        set_engine("glmnet"),
      grid       = 2,
      control    = control_grid(verbose = TRUE)
    )

ensemble_fit_glmnet

```

---

ensemble\_nested\_average

*Nested Ensemble Average*

---

### Description

Creates an Ensemble Model using Mean/Median Averaging in the Modeltime Nested Forecasting Workflow.

### Usage

```

ensemble_nested_average(
  object,
  type = c("mean", "median"),
  keep_submodels = TRUE,
  model_ids = NULL,
  control = control_nested_fit()
)

```

### Arguments

object	A nested modeltime object (inherits class nested_md1_time)
type	One of "mean" for mean averaging or "median" for median averaging
keep_submodels	Whether or not to keep the submodels in the nested modeltime table results
model_ids	A vector of id's (.model_id) identifying which submodels to use in the ensemble.
control	Controls various aspects of the ensembling process. See modeltime::control_nested_fit().

### Details

If we start with a nested modeltime table, we can add ensembles.

```
nested_modeltime_tbl

# Nested Modeltime Table
Trained on: .splits | Model Errors: [0]
# A tibble: 2 x 5
  id    .actual_data      .future_data      .splits      .modeltime_tables
  <fct> <list>              <list>            <list>       <list>
1 1_1   <tibble [104 x 2]> <tibble [52 x 2]> <split [52|52]> <mdl_time_tbl [2 x 5]>
2 1_3   <tibble [104 x 2]> <tibble [52 x 2]> <split [52|52]> <mdl_time_tbl [2 x 5]>
```

An ensemble can be added to a Nested modeltime table.

```
ensem <- nested_modeltime_tbl %>%
  ensemble_nested_average(
    type          = "mean",
    keep_submodels = TRUE,
    control       = control_nested_fit(allow_par = FALSE, verbose = TRUE)
  )
```

We can then verify the model has been added.

```
ensem %>% extract_nested_modeltime_table()
```

This produces an ensemble .model\_id 3, which is an ensemble of the first two models.

```
# A tibble: 4 x 6
  id    .model_id .model      .model_desc      .type .calibration_data
  <fct>   <dbl> <list>      <chr>            <chr> <list>
1 1_1     1 <workflow>  PROPHET           Test  <tibble [52 x 4]>
2 1_1     2 <workflow>  XGBOOST           Test  <tibble [52 x 4]>
3 1_1     3 <ensemble [2]> ENSEMBLE (MEAN): 2 MODELS Test  <tibble [52 x 4]>
```

Additional ensembles can be added by simply adding onto the nested modeltime table. Notice that we make use of model\_ids to make sure it only uses model id's 1 and 2.

```
ensem_2 <- ensem %>%
  ensemble_nested_average(
    type          = "median",
    keep_submodels = TRUE,
    model_ids     = c(1,2),
    control       = control_nested_fit(allow_par = FALSE, verbose = TRUE)
  )
```

This returns a 4th model that is a median ensemble of the first two models.

```
ensem_2 %>% extract_nested_modeltime_table()
# A tibble: 4 x 6
  id    .model_id .model      .model_desc      .type .calibration_data
```

	<fct>	<dbl>	<list>	<chr>	<chr>	<list>
1	1_1	1	<workflow>	PROPHET	Test	<tibble [52 x 4]>
2	1_1	2	<workflow>	XGBOOST	Test	<tibble [52 x 4]>
3	1_1	3	<ensemble [2]>	ENSEMBLE (MEAN): 2 MODELS	Test	<tibble [52 x 4]>
4	1_1	4	<ensemble [2]>	ENSEMBLE (MEDIAN): 2 MODELS	Test	<tibble [52 x 4]>

**Value**

The nested modeltime table with an ensemble model added.

---

ensemble\_nested\_weighted

*Nested Ensemble Weighted*

---

**Description**

Creates an Ensemble Model using Weighted Averaging in the Modeltime Nested Forecasting Workflow.

**Usage**

```
ensemble_nested_weighted(
  object,
  loadings,
  scale_loadings = TRUE,
  metric = "rmse",
  keep_submodels = TRUE,
  model_ids = NULL,
  control = control_nested_fit()
)
```

**Arguments**

object	A nested modeltime object (inherits class nested_md1_time)
loadings	A vector of weights corresponding to the loadings
scale_loadings	If TRUE, divides by the sum of the loadings to proportionally weight the submodels.
metric	The accuracy metric to rank models by the test accuracy table. Loadings are then applied in the order from best to worst models. Default: "rmse".
keep_submodels	Whether or not to keep the submodels in the nested modeltime table results
model_ids	A vector of id's (.model_id) identifying which submodels to use in the ensemble.
control	Controls various aspects of the ensembling process. See modeltime::control_nested_fit().



## Details

If we start with a nested modeltime table, we can add ensembles.

```
nested_modeltime_tbl

# Nested Modeltime Table
Trained on: .splits | Model Errors: [0]
# A tibble: 2 x 5
  id    .actual_data      .future_data      .splits      .modeltime_tables
  <fct> <list>              <list>           <list>       <list>
1 1_1  <tibble [104 x 2]> <tibble [52 x 2]> <split [52|52]> <mdl_time_tbl [2 x 5]>
2 1_3  <tibble [104 x 2]> <tibble [52 x 2]> <split [52|52]> <mdl_time_tbl [2 x 5]>
```

An ensemble can be added to a Nested modeltime table.

```
ensem <- nested_modeltime_tbl %>%
  ensemble_nested_weighted(
    loadings      = c(2,1),
    control       = control_nested_fit(allow_par = FALSE, verbose = TRUE)
  )
```

We can then verify the model has been added.

```
ensem %>% extract_nested_modeltime_table()
```

This produces an ensemble .model\_id 3, which is an ensemble of the first two models.

```
# A tibble: 4 x 6
  id    .model_id .model      .model_desc      .type .calibration_data
  <fct>   <dbl> <list>      <chr>            <chr> <list>
1 1_3     1 <workflow>  PROPHET          Test  <tibble [52 x 4]>
2 1_3     2 <workflow>  XGBOOST          Test  <tibble [52 x 4]>
3 1_3     3 <ensemble [2]> ENSEMBLE (WEIGHTED): 2 MODELS Test  <tibble [52 x 4]>
```

We can verify the loadings have been applied correctly. Note that the loadings will be applied based on the model with the lowest RMSE.

```
ensem %>%
  extract_nested_modeltime_table(1) %>%
  slice(3) %>%
  pluck(".model", 1)
```

Note that the xgboost model gets the 66% loading and prophet gets 33% loading. This is because xgboost has the lower RMSE in this case.

```

-- Modeltime Ensemble -----
  Ensemble of 2 Models (WEIGHTED)

# Modeltime Table
# A tibble: 2 x 6
  .model_id .model      .model_desc .type .calibration_data .loadings
    <int> <list>      <chr>      <chr> <list>             <dbl>
1       1 <workflow> PROPHET    Test <tibble [52 x 4]>  0.333
2       2 <workflow> XGBOOST   Test <tibble [52 x 4]>  0.667

```

**Value**

The nested modeltime table with an ensemble model added.

---

ensemble_weighted	<i>Creates a Weighted Ensemble Model</i>
-------------------	--

---

**Description**

Makes an ensemble by applying loadings to weight sub-model predictions

**Usage**

```
ensemble_weighted(object, loadings, scale_loadings = TRUE)
```

**Arguments**

object	A Modeltime Table
loadings	A vector of weights corresponding to the loadings
scale_loadings	If TRUE, divides by the sum of the loadings to proportionally weight the sub-models.

**Details**

The input to an ensemble\_weighted() model is always a Modeltime Table, which contains the models that you will ensemble.

**Weighting Method**

The weighted method uses uses loadings by applying a *loading x model prediction* for each sub-model.

**Value**

A mdl\_time\_ensemble object.

**Examples**

```
library(tidymodels)
library(modeltime)
library(modeltime.ensemble)
library(dplyr)
library(timetk)

# Make an ensemble from a Modeltime Table
ensemble_fit <- m750_models %>%
  ensemble_weighted(
    loadings = c(3, 3, 1),
    scale_loadings = TRUE
  )

ensemble_fit

# Forecast with the Ensemble
modeltime_table(
  ensemble_fit
) %>%
  modeltime_forecast(
    new_data = testing(m750_splits),
    actual_data = m750
  ) %>%
  plot_modeltime_forecast(
    .interactive = FALSE,
    .conf_interval_show = FALSE
  )
```

# Index

ensemble\_average, [2](#)  
ensemble\_model\_spec, [3](#)  
ensemble\_model\_spec(), [4](#)  
ensemble\_nested\_average, [6](#)  
ensemble\_nested\_weighted, [8](#)  
ensemble\_weighted, [10](#)