Package 'mlr3forecast'

August 24, 2026

Title Extending 'mlr3' to Time Series Forecasting

Version 0.2.0

Description Extends the 'mlr3' package and ecosystem to time series forecasting. Provides forecasting tasks, learners, resampling strategies, performance measures, and 'mlr3pipelines' operators for time-series feature engineering. Machine learning regression learners can be turned into forecasters through recursive and direct multi-step strategies.

License LGPL-3

URL https://mlr3forecast.mlr-org.com, https://github.com/mlr-org/mlr3forecast

BugReports https://github.com/mlr-org/mlr3forecast/issues

Depends mlr3 (>= 1.7.0), R (>= 3.6.0)

Imports backports (>= 1.5.0), checkmate (>= 2.0.0), cli, data.table (>= 1.18.0), generics (>= 0.1.2), lgr, mlr3misc (>= 0.22.0), mlr3pipelines (>= 0.11.0), paradox (>= 1.0.1), R6 (>= 2.4.1), stats, utils

Suggests distributional, fabletools, feasts, forecast (>= 9.0.2), ggplot2 (>= 3.4.0), greybox, mlr3tuning, nnet, nnfor (>= 0.9.9), prophet (>= 1.1.7), Rcatch22, Rlgt (>= 0.2.3), rpart, smooth (>= 4.4.0), testthat (>= 3.2.0), tidyselect, timeSeries, tsbox, tscount (>= 1.4.3), tsfeatures, tsibble, tsibbledata, vctrs, vdiffr (>= 1.0.0), withr (>= 3.0.0), xts, zoo

Config/roxygen2/markdown TRUE

Config/roxygen2/r6 TRUE

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Collate 'DirectForecaster.R' 'LearnerFcst.R' 'zzz.R'
'LearnerFcstAdam.R' 'LearnerFcstAr.R' 'LearnerFcstArfima.R'
'LearnerFcstArima.R' 'LearnerFcstAutoAdam.R'
'LearnerFcstAutoArima.R' 'LearnerFcstAutoCes.R'
'LearnerFcstAutoGum.R' 'LearnerFcstAutoMsarima.R'
'LearnerFcstAutoSsarima.R' 'LearnerFcstBaggedModel.R'
'LearnerFcstBats.R' 'LearnerFcstCes.R' 'LearnerFcstCroston.R'
'LearnerFcstElm.R' 'LearnerFcstEs.R' 'LearnerFcstEts.R'
'LearnerFcstForecast.R' 'LearnerFcstGum.R'
'LearnerFcstHoltWinters.R' 'LearnerFcstMean.R'
'LearnerFcstMlp.R' 'LearnerFcstMsarima.R' 'LearnerFcstNnetar.R'
'LearnerFcstProphet.R' 'LearnerFcstRandomWalk.R'
'LearnerFcstRlgt.R' 'LearnerFcstSma.R' 'LearnerFcstSmooth.R'
'LearnerFcstSparma.R' 'LearnerFcstSpline.R'
'LearnerFcstSsarima.R' 'LearnerFcstStlm.R'
'LearnerFcstStructTS.R' 'LearnerFcstTbats.R'
'LearnerFcstTheta.R' 'LearnerFcstTscount.R' 'LearnerFcstTslm.R'
'MeasureACF1.R' 'MeasureCoverage.R' 'MeasureDirectional.R'
'MeasureMPE.R' 'MeasureMSIS.R' 'MeasurePinball.R'
'MeasureScaled.R' 'MeasureWAPE.R' 'MeasureWinkler.R'
'PipeOpFcstAvg.R' 'PipeOpFcstCatch22.R' 'PipeOpFcstFeasts.R'
'PipeOpFcstFourier.R' 'PipeOpFcstLags.R' 'PipeOpFcstRolling.R'
'PipeOpFcstSplitKey.R' 'PipeOpFcstTsfeats.R'
'PipeOpFcstUniteKey.R' 'PipeOpTargetTrafo.R'
'PipeOpTargetTrafoBoxCox.R' 'PredictionDataFcst.R'
'PredictionFcst.R' 'RecursiveForecaster.R' 'ResamplingFcstCV.R'
'ResamplingFcstHoldout.R' 'TaskFcst.R'
'TaskFcstAirpassengers.R' 'TaskFcstElectricity.R'
'TaskFcstLivestock.R' 'TaskFcstLynx.R' 'TaskFcstUsaccdeaths.R'
'as_task_fcst.R' 'assertions.R' 'autoplot.R' 'bibentries.R'
'direct_forecaster.R' 'forecast.R' 'helper.R'
'helper_data_table.R' 'helper_freq.R' 'helper_key.R'
'partition.R' 'pipeline_fcst_local.R' 'recursive_forecaster.R'
'reexports.R' 'selector.R' 'tsf.R'

NeedsCompilation no

Author Maximilian Mücke [aut, cre] (ORCID:
<https://orcid.org/0009-0000-9432-9795>),
Marc Becker [aut] (ORCID: <https://orcid.org/0000-0002-8115-0400>),
Bernd Bischl [aut] (ORCID: <https://orcid.org/0000-0001-6002-6980>)

Maintainer Maximilian Mücke <muecke.maximilian@gmail.com>

Repository CRAN

Date/Publication 2026-08-24 10:10:08 UTC

Contents

`mlr3forecast-package` *mlr3forecast: Extending 'mlr3' to Time Series Forecasting*

Description

Extends the 'mlr3' package and ecosystem to time series forecasting. Provides forecasting tasks, learners, resampling strategies, performance measures, and 'mlr3pipelines' operators for time-series feature engineering. Machine learning regression learners can be turned into forecasters through recursive and direct multi-step strategies.

Author(s)

Maintainer: Maximilian Mücke <muecke.maximilian@gmail.com> (ORCID)

Authors:

- Maximilian Mücke <muecke.maximilian@gmail.com> (ORCID)
- Marc Becker <marcbecker@posteo.de> (ORCID)
- Bernd Bischl <bernd.bischl@gmail.com> (ORCID)

See Also

Useful links:

- https://mlr3forecast.mlr-org.com
- https://github.com/mlr-org/mlr3forecast
- Report bugs at https://github.com/mlr-org/mlr3forecast/issues

---

as_task_fcst *Convert to a Forecast Task*

Description

Convert object to a TaskFcst. This is a S3 generic. mlr3forecast ships with methods for the following objects:

1. TaskFcst: ensure the identity
2. `data.frame()` and mlr3::DataBackend: provides an alternative to the constructor of TaskFcst.
3. `ts`: from base R time series objects (univariate and multivariate).
4. `zoo` and `xts`: from zoo/xts time series objects.
5. `timeSeries`: from Rmetrics timeSeries objects.
6. `tsf`: from tsf format data.
7. `tbl_ts`: from tsibble objects.

Usage

```
as_task_fcst(x, ...)

as_tasks_fcst(x, ...)

## Default S3 method:
as_tasks_fcst(x, ...)

## S3 method for class 'list'
as_tasks_fcst(x, ...)

## S3 method for class 'TaskFcst'
as_task_fcst(x, clone = FALSE, ...)

## S3 method for class 'DataBackend'
as_task_fcst(
  x,
  target,
  order,
  key = character(),
  freq = NULL,
  id = deparse1(substitute(x)),
  label = NA_character_,
  ...
)

## S3 method for class 'data.frame'
as_task_fcst(
  x,
  target,
  order,
  key = character(),
  freq = NULL,
  id = deparse1(substitute(x)),
  label = NA_character_,
  ...
)

## S3 method for class 'tsf'
as_task_fcst(x, id = deparse1(substitute(x)), label = NA_character_, ...)

## S3 method for class 'ts'
as_task_fcst(
  x,
  freq = NULL,
  id = deparse1(substitute(x)),
  label = NA_character_,
  ...
```

```
)

## S3 method for class 'zoo'
as_task_fcst(
  x,
  freq = NULL,
  id = deparse1(substitute(x)),
  label = NA_character_,
  ...
)

## S3 method for class 'timeSeries'
as_task_fcst(
  x,
  freq = NULL,
  id = deparse1(substitute(x)),
  label = NA_character_,
  ...
)

## S3 method for class 'tbl_ts'
as_task_fcst(
  x,
  target,
  freq = NULL,
  id = deparse1(substitute(x)),
  label = NA_character_,
  ...
)
```

Arguments

`x`	(any) Object to convert.
`...`	(any) Additional arguments.
`clone`	(`logical(1)`) If `TRUE`, ensures that the returned object is not the same as the input `x`.
`target`	(`character(1)`) Name of the target column.
`order`	(`character(1)`) Name of the order column.
`key`	(`character()`) Names of the columns whose combination identifies a series. Key columns must be character, integer, factor, or ordered columns.
`freq`	(`character(1)` | `numeric(1)` | `NULL`) Frequency of the time series. A `seq()`-compatible string gives the calendar step

of the time grid, e.g. `"1 month"`, `"day"`, `"3 months"`, `"1 hour"`, `"week"`. A positive number gives the seasonal period (as in `stats::ts()`) for an integer or numeric order column; the grid step is then inferred from the data.

`id` (`character(1)`)
Id for the new task. Defaults to the (deparsed and substituted) name of the data argument.

`label` (`character(1)`)
Label for the new instance.

Value

TaskFcst.

Examples

```
library(data.table)
airpassengers = tsbox::ts_dt(AirPassengers)
setnames(airpassengers, c("month", "passengers"))
as_task_fcst(airpassengers, target = "passengers", order = "month", freq = "month")
```

`autoplot.PredictionFcst`
Plot for Forecast Predictions

Description

Generates a forecast plot for PredictionFcst. The point forecast is drawn over the time index. When a `task` is supplied, the historical series is overlaid and the forecast region is drawn in a distinct colour, connected to the last historical observation for visual continuity.

For quantile forecasts, symmetric quantile pairs (e.g. the 10% and 90% quantiles) are drawn as shaded central prediction interval ribbons over the forecast region, shaded darker for narrower intervals and labelled by their level (e.g. 80, 95) in a legend. Quantiles without a symmetric partner are not drawn.

Usage

```
## S3 method for class 'PredictionFcst'
autoplot(
  object,
  task = NULL,
  theme = ggplot2::theme_minimal(),
  facets = FALSE,
  ...
)
```

Arguments

object	(PredictionFcst).
task	(TaskFcst | NULL) Optional task providing the historical series to overlay. When NULL, only the forecast region is drawn.
theme	(ggplot2::theme()) The ggplot2::theme_minimal() is applied by default to all plots.
facets	(logical(1)) For keyed tasks, draw one panel per series instead of one coloured line per series. Default FALSE.
...	(any) Additional argument, passed down to the underlying geom or plot functions.

Value

ggplot2::ggplot() object.

Examples

```
task = tsk("airpassengers")
learner = lrn("fcst.auto_arima")$train(task)
p = forecast(learner, task, h = 12)
ggplot2::autoplot(p, task = task)
```

---

autoplot.TaskFcst *Plot for Forecast Tasks*

Description

Generates plots for TaskFcst.

Usage

```
## S3 method for class 'TaskFcst'
autoplot(object, theme = ggplot2::theme_minimal(), facets = FALSE, ...)
```

Arguments

object	(TaskFcst).
theme	(ggplot2::theme()) The ggplot2::theme_minimal() is applied by default to all plots.
facets	(logical(1)) For keyed tasks, draw one panel per series instead of one coloured line per series. Default FALSE.
...	(any) Additional argument, passed down to the underlying geom or plot functions.

Value

ggplot2::ggplot() object.

Examples

```
task = tsk("airpassengers")
ggplot2::autoplot(task)
```

DirectForecaster *Direct Multi-Step Forecast Learner*

Description

Trains a separate model for each forecast horizon. For horizon `h` with base lags `1:p`, model `h` uses lags `h:(h+p-1)`, so that at prediction time only observed values are needed. Unlike RecursiveForecaster, predictions do not feed back into subsequent steps (no error accumulation).

Lag features are managed internally via `lags`. Do not add iterative feature PipeOps (property `"fcst_iterative"`, e.g. PipeOpFcstLags, PipeOpFcstRolling), which are rejected at construction.

Super class

mlr3::Learner -> DirectForecaster

Active bindings

`learner` (mlr3::Learner)
The base regression learner.

`native_model` (named `list()`)
The fitted models.

`lags` (`integer()`)
The base lags.

`horizons` (`integer()`)
The forecast horizons.

`param_set` (paradox::ParamSet)
Set of hyperparameters.

`marshaled` (`logical(1)`)
Whether the learner's model is currently in marshaled form.

`predict_type` (`character(1)`)
Stores the currently active predict type.

Methods

Public methods:

- DirectForecaster$new()
- DirectForecaster$print()
- DirectForecaster$marshal()
- DirectForecaster$unmarshal()
- DirectForecaster$importance()
- DirectForecaster$selected_features()
- DirectForecaster$oob_error()
- DirectForecaster$clone()

`DirectForecaster$new()`: Creates a new instance of this R6 class.

Usage:

```
DirectForecaster$new(
  learner,
  lags,
  horizons,
  id = NULL,
  param_vals = list(),
  predict_type = NULL
)
```

Arguments:

`learner` (mlr3::Learner | mlr3pipelines::Graph | mlr3pipelines::PipeOp)
: A regression learner or a graph/PipeOp (without PipeOpFcstLags).

`lags` (`integer()`)
: The base lag values. Exposed in `$param_set` as `lags`, so it can be tuned via mlr3tuning::AutoTuner.

`horizons` (`integer()`)
: Either a single integer `H` (expanded to `1:H`) or an integer vector of specific horizons. One model is trained per horizon. At predict time each test row is routed to the model matching its step-distance from the end of training, so with specific horizons (e.g. `c(2L, 4L, 6L)`) the test set may only contain rows at those exact steps ahead.

`id` (`character(1)` | `NULL`)
: Identifier, default `NULL` (auto-generated from the learner id).

`param_vals` (named `list()`)
: Hyperparameter values applied to every horizon model. Per-horizon hyperparameters are not currently supported.

`predict_type` (`character(1)` | `NULL`)
: The predict type, default `NULL`.

`DirectForecaster$print()`: Printer.

Usage:

```
DirectForecaster$print(...)
```

Arguments:

`...` (ignored).

`DirectForecaster$marshal()`: Marshal the learner's model.

Usage:

```
DirectForecaster$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`DirectForecaster$unmarshal()`: Unmarshal the learner's model.

Usage:

```
DirectForecaster$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`DirectForecaster$importance()`: The importance scores of the base learner of each horizon model, if it supports them. Unlike mlr3::Learner's `$importance()` this returns one vector per horizon, so the `"importance"` property is deliberately not advertised.

Usage:

```
DirectForecaster$importance()
```

Returns: Named `list()` of named `numeric()`, one element per horizon (`h1`, `h2`, ...).

`DirectForecaster$selected_features()`: The selected features of the base learner of each horizon model, if it supports them.

Usage:

```
DirectForecaster$selected_features()
```

Returns: Named `list()` of `character()`, one element per horizon (`h1`, `h2`, ...).

`DirectForecaster$oob_error()`: The out-of-bag error of the base learner of each horizon model, if it supports it.

Usage:

```
DirectForecaster$oob_error()
```

Returns: Named `list()` of `numeric(1)`, one element per horizon (`h1`, `h2`, ...).

`DirectForecaster$clone()`: The objects of this class are cloneable with this method.

Usage:

```
DirectForecaster$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3pipelines)

task = tsk("airpassengers")
split = partition(task, ratio = 0.8)

# simple: one model per horizon
flrn = DirectForecaster$new(lrn("regr.rpart"), lags = 1:3, horizons = length(split$test))
flrn$train(task, split$train)
flrn$predict(task, split$test)

# or use the direct_forecaster() helper
flrn = direct_forecaster(lrn("regr.rpart"), lags = 1:3, horizons = length(split$test))
flrn$train(task, split$train)
flrn$predict(task, split$test)
```

---

direct_forecaster *Create a Direct Forecast Learner*

Description

Function to create a DirectForecaster object. This is the recommended way to construct a direct forecaster. It is a thin wrapper around `DirectForecaster$new()`.

A direct forecaster trains a separate regression model per forecast horizon, so predictions never feed back into one another (no error accumulation). For the recursive strategy (a single iterated model) see `recursive_forecaster()`.

Usage

```
direct_forecaster(
  learner,
  lags,
  horizons,
  id = NULL,
  param_vals = list(),
  predict_type = NULL
)
```

Arguments

`learner`	(mlr3::Learner | mlr3pipelines::Graph | mlr3pipelines::PipeOp) A regression learner or a graph/PipeOp (without PipeOpFcstLags).
`lags`	(`integer()`) The base lag values. Exposed in `$param_set` as `lags`, so it can be tuned via mlr3tuning::AutoTuner.

horizons	(integer()) Either a single integer H (expanded to 1:H) or an integer vector of specific horizons. One model is trained per horizon.
id	(character(1) | NULL) Identifier, default NULL (auto-generated from the learner id).
param_vals	(named list()) Hyperparameter values applied to every horizon model. Per-horizon hyperparameters are not currently supported.
predict_type	(character(1) | NULL) The predict type, default NULL.

Value

DirectForecaster.

Examples

```
library(mlr3pipelines)

task = tsk("airpassengers")
split = partition(task, ratio = 0.8)

# one model per horizon
flrn = direct_forecaster(lrn("regr.rpart"), lags = 1:3, horizons = length(split$test))
flrn$train(task, split$train)
flrn$predict(task, split$test)
```

---

download_zenodo_record

Download tsf file from Zenodo

Description

Downloads a tsf file from Zenodo using the provided record ID and dataset name.

Usage

```
download_zenodo_record(record_id = 4656222, dataset_name = "m3_yearly_dataset")
```

Arguments

record_id	(integer(1)) The Zenodo record ID.
dataset_name	(character(1)) The name of the dataset to download.

Value

(data.table::data.table()) with class "tsf". If the file contains a frequency or horizon, the "frequency" and "horizon" attributes are set, respectively.

References

Godahewa R, Bergmeir C, Webb GI, Hyndman RJ, Montero-Manso P (2021). "Monash time series forecasting archive." *arXiv preprint arXiv:2105.06643*.

Examples

```
## Not run:
library(data.table)
dt = download_zenodo_record(record_id = 4656222, dataset_name = "m3_yearly_dataset")

# optional renaming
setnames(dt, c("id", "date", "value"))

# transform into single task
task = as_task_fcst(dt)

# or split up for forecast learners that don't allow key columns
tasks = as_tasks_fcst(split(dt, by = "id", keep.by = FALSE))

# benchmark
learners = lrns(c("fcst.auto_arima", "fcst.ets", "fcst.random_walk"))
resampling = rsmp("fcst.holdout", ratio = 0.8)
design = benchmark_grid(tasks, learners, resampling)
bmr = benchmark(design)
bmr$aggregate(msr("regr.rmse"))[, .(rmse = mean(regr.rmse)), by = learner_id]

## End(Not run)
```

---

forecast.Learner *Forecast from a Trained Learner*

Description

Generates h future rows from the task's skeleton (using generate_newdata()), optionally overlays user-supplied newdata onto those rows, and predicts with the trained learner via mlr3::Learner$predict_newdata(). Works with RecursiveForecaster, DirectForecaster, and classic LearnerFcst* forecasters.

Usage

```
## S3 method for class 'Learner'
forecast(object, task, h = 12L, newdata = NULL, ...)
```

Arguments

`object`	(mlr3::Learner) A trained forecast learner.
`task`	(TaskFcst) Provides the metadata needed to construct future rows: the order column (to extend the time index), key columns (for keyed tasks), `freq`, and the column-type schema expected by `predict_newdata()`. The task's data values are not used. Pass the training task or any other schema-compatible TaskFcst.
`h`	(`integer(1)`) Forecast horizon — number of future time steps per key.
`newdata`	(`data.frame()` | `NULL`) Optional exogenous features for future rows. Must contain the order column (and any key columns for keyed tasks), and every row must match a row of the generated future grid. Columns other than those are overlaid onto the generated skeleton, while skeleton rows without a match keep `NA`.
`...`	(any) Must be empty.

Value

mlr3::Prediction.

`generate_newdata` *Generate new data for a forecast task*

Description

Generate new data for a forecast task

Usage

```
generate_newdata(task, n = 1L)
```

Arguments

`task`	TaskFcst Task.
`n`	(`integer(1)`) Number of new data points to generate. Default `1L`.

Details

Future dates are extrapolated by stepping the order column. For calendar `freq` (`month`/`quarter`/`year`), the origin's day-of-month is carried forward and clamped to each target month's last valid day. Other freqs use `base::seq()`. Month-end is not inferred, so use a first-of-month or period-style index for genuine month-end series.

Value

A data.table::data.table() with n new data points.

`LearnerFcst` *Forecast Learner*

Description

This Learner specializes mlr3::LearnerRegr for forecast problems:

- `task_type` is set to `"fcst"`.
- Creates mlr3::Predictions of class PredictionFcst.
- Possible values for `predict_types` are:
  - `"response"`: Predicts a numeric response for each observation in the test set.
  - `"se"`: Predicts the standard error for each value of response for each observation in the test set.
  - `"distr"`: Probability distribution as `VectorDistribution` object (requires package `distr6`, available via repository `https://raphaels1.r-universe.dev`).
  - `"quantiles"`: Predicts quantile estimates for each observation in the test set. Set `$quantiles` to specify the quantiles to predict and `$quantile_response` to specify the response quantile. See the mlr3book on quantile regression for more details.

Predefined learners can be found in the dictionary mlr3::mlr_learners.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst`

Active bindings

`native_model` (any)
The fitted model.

Methods

Public methods:

- `LearnerFcst$new()`
- `LearnerFcst$clone()`

`LearnerFcst$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcst$new(
  id,
  param_set = ps(),
  predict_types = "response",
  feature_types = character(),
  properties = character(),
  packages = character(),
  label = NA_character_,
  man = NA_character_
)
```

Arguments:

`id (character(1))`
: Identifier for the new instance.

`param_set (paradox::ParamSet)`
: Set of hyperparameters.

`predict_types (character())`
: Supported predict types. Must be a subset of `mlr_reflections$learner_predict_types`.

`feature_types (character())`
: Feature types the learner operates on. Must be a subset of `mlr_reflections$task_feature_types`.

`properties (character())`
: Set of properties of the mlr3::Learner. Must be a subset of `mlr_reflections$learner_properties`. The following properties are currently standardized and understood by learners in **mlr3**:
  - `"missings"`: The learner can handle missing values in the data.
  - `"weights"`: The learner supports observation weights.
  - `"offset"`: The learner can incorporate offset values to adjust predictions.
  - `"importance"`: The learner supports extraction of importance scores, i.e. comes with an `$importance()` extractor function (see section on optional extractors in mlr3::Learner).
  - `"selected_features"`: The learner supports extraction of the set of selected features, i.e. comes with a `$selected_features()` extractor function (see section on optional extractors in mlr3::Learner).
  - `"oob_error"`: The learner supports extraction of estimated out of bag error, i.e. comes with a `oob_error()` extractor function (see section on optional extractors in mlr3::Learner).
  - `"validation"`: The learner can use a validation task during training.
  - `"internal_tuning"`: The learner is able to internally optimize hyperparameters (those are also tagged with `"internal_tuning"`).
  - `"marshal"`: To save learners with this property, you need to call `$marshal()` first. If a learner is in a marshaled state, you call first need to call `$unmarshal()` to use its model, e.g. for prediction.
  - `"hotstart_forward"`: The learner supports to hotstart a model forward.
  - `"hotstart_backward"`: The learner supports hotstarting a model backward.
  - `"featureless"`: The learner does not use features.

`packages (character())`
: Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via `requireNamespace()`.

`label (character(1))`
: Label for the new instance.

`man (character(1))`
String in the format `[pkg]::[topic]` pointing to a manual page for this object. The referenced help package can be opened via method `$help()`.

`LearnerFcst$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcst$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarima`, `mlr_learners_fcst.nnetar`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.random_walk`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.stlm`, `mlr_learners_fcst.struct_t`, `mlr_learners_fcst.tbats`, `mlr_learners_fcst.theta`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# get all forecast learners from mlr_learners:
lrns = mlr_learners$mget(mlr_learners$keys("^fcst"))
names(lrns)

# get a specific learner from mlr_learners:
mlr_learners$get("fcst.ets")
lrn("fcst.auto_arima")
```

`mlr_graphs_fcst.local` *Create a Graph to Fit Local Per-Series Forecast Models*

Description

Create a new Graph that wraps `graph` between `po("fcst.splitkey")` and `po("fcst.unitekey")`, fitting one local model per series of a keyed TaskFcst instead of one global model pooled across series.

All input arguments are cloned and have no references in common with the returned Graph.

Usage

```
pipeline_fcst_local(graph, key = "key")
```

Arguments

`graph`	(any) Object coercible to a Graph with `mlr3pipelines::as_graph()`, being wrapped between `po("fcst.splitkey")` and `po("fcst.unitekey")`. The graph should return `NULL` during training and a PredictionFcst during prediction.
`key`	(`character(1)`) Name of the rebuilt series-identity column in the united prediction's `extra` slot, default `"key"`. Set it to the task's key column name to get predictions column-compatible with global forecasters such as RecursiveForecaster.

Value

Graph

Examples

```
library(mlr3pipelines)
library(data.table)
dt = CJ(
  month = seq(as.Date("2024-01-01"), by = "month", length.out = 36L),
  id = factor(c("a", "b"))
)
dt[, value := rnorm(.N, mean = fifelse(id == "a", 10, 20))]
task = as_task_fcst(dt, target = "value", order = "month", key = "id", freq = "month")
flrn = as_learner(ppl("fcst.local", lrn("fcst.ets")))$train(task)
forecast(flrn, task, 12L)
```

`mlr_learners_fcst.adam`

ADAM Forecast Learner

Description

Augmented Dynamic Adaptive Model (ADAM) Forecast Learner model. Calls `smooth::adam()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.adam")
lrn("fcst.adam")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels
model	untyped	"ZXZ"	
lags	untyped	-	
orders	untyped	list(ar = 0, i = 0, ma = 0, select = FALSE)	
constant	logical	FALSE	TRUE, FALSE
regressors	character	use	use, select, adapt
occurrence	character	none	none, auto, fixed, general, odds-ratio, inverse-odds-rati
distribution	character	default	default, dnorm, dlaplace, ds, dgnorm, dlnorm, dinvgaus
loss	character	likelihood	likelihood, MSE, MAE, HAM, LASSO, RIDGE, MSE
outliers	character	ignore	ignore, use, select
holdout	logical	FALSE	TRUE, FALSE
persistence	untyped	NULL	
phi	numeric	NULL	
initial	character	backcasting	backcasting, optimal, two-stage, complete
arma	untyped	NULL	
ic	character	AICc	AICc, AIC, BIC, BICc
bounds	character	usual	usual, admissible, none
silent	logical	TRUE	TRUE, FALSE
ets	character	conventional	conventional, adam

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstAdam`

Methods

Public methods:

- `LearnerFcstAdam$new()`
- `LearnerFcstAdam$clone()`

`LearnerFcstAdam$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstAdam$new()
```

`LearnerFcstAdam$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstAdam$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

Svetunkov I (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)*, 1st edition. Chapman and Hall/CRC. doi:10.1201/9781003452652. https://openforecast.org/adam/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:

  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarir mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.adam")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.ar *Autoregressive Forecast Learner*

Description

Univariate autoregressive model with the order selected by AIC. Calls stats::ar() from package **stats** and forecasts via forecast::forecast().

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.ar")
lrn("fcst.ar")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
aic	logical	TRUE	TRUE, FALSE	-
order.max	integer	NULL		$[1, \infty)$
method	character	yule-walker	yule-walker, burg, ols, mle, yw	-
demean	logical	TRUE	TRUE, FALSE	-
var.method	integer	1		$[1, 2]$
intercept	logical	-	TRUE, FALSE	-
simulate	logical	FALSE	TRUE, FALSE	-
bootstrap	logical	FALSE	TRUE, FALSE	-
innov	untyped	NULL		-
npaths	integer	5000		$[1, \infty)$
lambda	untyped	NULL		-
biasadj	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstAr`

Methods

Public methods:

- `LearnerFcstAr$new()`
- `LearnerFcstAr$clone()`

`LearnerFcstAr$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstAr$new()
```

`LearnerFcstAr$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstAr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Brockwell PJ, Davis RA (1991). *Time Series: Theory and Methods*, 2nd edition. Springer, New York.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarir mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.ar")
print(learner)
```

```
# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.arfima` *ARFIMA Forecast Learner*

Description

ARFIMA model. Calls `forecast::arfima()` from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.arfima")
lrn("fcst.arfima")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
drange	untyped	c(0, 0.5)		-
estim	character	mle	mle, ls	-
lambda	untyped	NULL		-
biasadj	logical	FALSE	TRUE, FALSE	-
simulate	logical	FALSE	TRUE, FALSE	-
bootstrap	logical	FALSE	TRUE, FALSE	-
npaths	integer	5000		$[1, \infty)$
max.p	integer	5		$[0, \infty)$
max.q	integer	5		$[0, \infty)$
max.order	integer	5		$[0, \infty)$
start.p	integer	2		$[0, \infty)$
start.q	integer	2		$[0, \infty)$
ic	character	aicc	aicc, aic, bic	-
stepwise	logical	TRUE	TRUE, FALSE	-
nmodels	integer	94		$[0, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
approximation	logical	-	TRUE, FALSE	-
method	character	NULL	CSS-ML, ML, CSS	-
truncate	integer	NULL		$[1, \infty)$
parallel	logical	FALSE	TRUE, FALSE	-
num.cores	integer	2		$[1, \infty)$
transform.pars	logical	TRUE	TRUE, FALSE	-
fixed	untyped	NULL		-
init	untyped	NULL		-
SSinit	character	Gardner1980	Gardner1980, Rossignol2011	-
n.cond	integer	-		$[1, \infty)$
optim.method	character	BFGS	Nelder-Mead, BFGS, CG, L-BFGS-B, SANN, Brent	-
optim.control	untyped	list()		-
kappa	numeric	1e+06		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstArfima`

Methods

Public methods:

- `LearnerFcstArfima$new()`
- `LearnerFcstArfima$clone()`

`LearnerFcstArfima$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstArfima$new()
```

`LearnerFcstArfima$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstArfima$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Haslett J, Raftery AE (1989). "Space-time Modelling with Long-memory Dependence: Assessing Ireland's Wind Power Resource." *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, **38**(1), 1–21.

Hyndman RJ, Khandakar Y (2008). "Automatic Time Series Forecasting: The forecast Package for R." *Journal of Statistical Software*, **27**(3), 1–22. doi:10.18637/jss.v027.i03.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarir` `mlr_learners_fcst.nnetar`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.random_walk`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.stlm`, `mlr_learners_fcst.struct_ts`, `mlr_learners_fcst.tbats` `mlr_learners_fcst.theta`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.arfima")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.arima

ARIMA Forecast Learner

Description

Autoregressive Integrated Moving Average Forecast (ARIMA) model. Calls `forecast::Arima()` from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.arima")
lrn("fcst.arima")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
order	untyped	c(0L, 0L, 0L)		-
seasonal	untyped	c(0L, 0L, 0L)		-
include.mean	logical	TRUE	TRUE, FALSE	-
include.drift	logical	FALSE	TRUE, FALSE	-
include.constant	logical	NULL	TRUE, FALSE	-
lambda	untyped	NULL		-
biasadj	logical	-	TRUE, FALSE	-
method	character	CSS-ML	CSS-ML, ML, CSS	-
simulate	logical	FALSE	TRUE, FALSE	-
bootstrap	logical	FALSE	TRUE, FALSE	-
npaths	integer	5000		$[1, \infty)$
transform.pars	logical	TRUE	TRUE, FALSE	-
fixed	untyped	NULL		-
init	untyped	NULL		-
SSinit	character	Gardner1980	Gardner1980, Rossignol2011	-
n.cond	integer	-		$[1, \infty)$
optim.method	character	BFGS	Nelder-Mead, BFGS, CG, L-BFGS-B, SANN, Brent	-
optim.control	untyped	list()		-
kappa	numeric	1e+06		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstArima`

Methods

Public methods:

- `LearnerFcstArima$new()`
- `LearnerFcstArima$clone()`

`LearnerFcstArima$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstArima$new()
```

`LearnerFcstArima$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstArima$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Hyndman RJ, Athanasopoulos G (2018). *Forecasting: principles and practice*, 2nd edition. OTexts, Melbourne, Australia. https://OTexts.com/fpp2/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarir mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.arima")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.auto_adam` *Auto ADAM Forecast Learner*

Description

Auto Augmented Dynamic Adaptive Model (ADAM) model. Calls `smooth::auto.adam()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.auto_adam")
lrn("fcst.auto_adam")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels
model	untyped	"ZXZ"	
lags	untyped	-	
orders	untyped	list(ar = c(3, 3), i = c(2, 1), ma = c(3, 3), select = TRUE)	
regressors	character	use	use, select, adapt
occurrence	character	none	none, auto, fixed, general, odds-ra
distribution	untyped	c("dnorm", "dlaplace", "ds", "dgnorm", "dlnorm", "dinvgauss",	

outliers	character	ignore	ignore, use, select
holdout	logical	FALSE	TRUE, FALSE
persistence	untyped	NULL	
phi	numeric	NULL	
initial	character	backcasting	backcasting, optimal, two-stage, c
arma	untyped	NULL	
ic	character	AICc	AICc, AIC, BIC, BICc
bounds	character	usual	usual, admissible, none
silent	logical	TRUE	TRUE, FALSE
parallel	logical	FALSE	TRUE, FALSE
ets	character	conventional	conventional, adam

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstAutoAdam`

Methods

Public methods:

- `LearnerFcstAutoAdam$new()`
- `LearnerFcstAutoAdam$clone()`

`LearnerFcstAutoAdam$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstAutoAdam$new()
```

`LearnerFcstAutoAdam$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstAutoAdam$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

Svetunkov I (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)*, 1st edition. Chapman and Hall/CRC. doi:10.1201/9781003452652. https://openforecast.org/adam/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarir mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.auto_adam")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.auto_arima` *Auto ARIMA Forecast Learner*

Description

Auto ARIMA model. Calls `forecast::auto.arima()` from package **forecast**.

Parallel fitting via `parallel` or `mlr3::set_threads()` requires `stepwise = FALSE`; with the default stepwise search, `forecast::auto.arima()` warns and fits in serial.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.auto_arima")
lrn("fcst.auto_arima")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
d	integer	NA		$[0, \infty)$
D	integer	NA		$[0, \infty)$
max.p	integer	5		$[0, \infty)$
max.q	integer	5		$[0, \infty)$
max.P	integer	2		$[0, \infty)$
max.Q	integer	2		$[0, \infty)$
max.order	integer	5		$[0, \infty)$
max.d	integer	2		$[0, \infty)$
max.D	integer	1		$[0, \infty)$

start.p	integer	2		$[0, \infty)$
start.q	integer	2		$[0, \infty)$
start.P	integer	1		$[0, \infty)$
start.Q	integer	1		$[0, \infty)$
stationary	logical	FALSE	TRUE, FALSE	-
seasonal	logical	TRUE	TRUE, FALSE	-
ic	character	aicc	aicc, aic, bic	-
stepwise	logical	TRUE	TRUE, FALSE	-
nmodels	integer	94		$[0, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
approximation	logical	-	TRUE, FALSE	-
method	character	NULL	CSS-ML, ML, CSS	-
truncate	integer	NULL		$[1, \infty)$
test	character	kpss	kpss, adf, pp	-
test.args	untyped	list()		-
seasonal.test	character	seas	seas, ocsb, hegy, ch	-
seasonal.test.args	untyped	list()		-
allowdrift	logical	TRUE	TRUE, FALSE	-
allowmean	logical	TRUE	TRUE, FALSE	-
biasadj	logical	FALSE	TRUE, FALSE	-
parallel	logical	FALSE	TRUE, FALSE	-
num.cores	integer	2		$[1, \infty)$
lambda	untyped	NULL		-
simulate	logical	FALSE	TRUE, FALSE	-
bootstrap	logical	FALSE	TRUE, FALSE	-
npaths	integer	5000		$[1, \infty)$
transform.pars	logical	TRUE	TRUE, FALSE	-
fixed	untyped	NULL		-
init	untyped	NULL		-
SSinit	character	Gardner1980	Gardner1980, Rossignol2011	-
n.cond	integer	-		$[1, \infty)$
optim.method	character	BFGS	Nelder-Mead, BFGS, CG, L-BFGS-B, SANN, Brent	-
optim.control	untyped	list()		-
kappa	numeric	1e+06		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstAutoArima`

Methods

Public methods:

- `LearnerFcstAutoArima$new()`
- `LearnerFcstAutoArima$clone()`

`LearnerFcstAutoArima$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstAutoArima$new()
```

`LearnerFcstAutoArima$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstAutoArima$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Hyndman RJ, Khandakar Y (2008). "Automatic Time Series Forecasting: The forecast Package for R." *Journal of Statistical Software*, **27**(3), 1–22. doi:10.18637/jss.v027.i03.

Wang X, Smith K, Hyndman R (2006). "Characteristic-based clustering for time series data." *Data Mining and Knowledge Discovery*, **13**, 335–364.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarir` `mlr_learners_fcst.nnetar`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.random_walk`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.stlm`, `mlr_learners_fcst.struct_ts`, `mlr_learners_fcst.tbats` `mlr_learners_fcst.theta`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.auto_arima")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.auto_ces

Auto CES Forecast Learner

Description

Auto Complex Exponential Smoothing (CES) model. Calls `smooth::auto.ces()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.auto_ces")
lrn("fcst.auto_ces")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels
seasonality	character	none	none, simple, partial, full
lags	untyped	-	
initial	character	backcasting	backcasting, optimal, two-stage, complete
ic	character	AICc	AICc, AIC, BIC, BICc
loss	character	likelihood	likelihood, MSE, MAE, HAM, MSEh, TMSE, GTMSE, MSCE, GPL
holdout	logical	FALSE	TRUE, FALSE
bounds	character	admissible	admissible, none
silent	logical	TRUE	TRUE, FALSE
regressors	character	use	use, select, adapt

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstAutoCes`

Methods

Public methods:

- `LearnerFcstAutoCes$new()`
- `LearnerFcstAutoCes$clone()`

`LearnerFcstAutoCes$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstAutoCes$new()
```

`LearnerFcstAutoCes$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstAutoCes$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

Svetunkov I (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)*, 1st edition. Chapman and Hall/CRC. doi:10.1201/9781003452652. https://openforecast.org/adam/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarir mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.auto_ces")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.auto_gum`
Auto GUM Forecast Learner

Description

Automatic selection over Generalised Univariate Model (GUM) specifications via an information criterion. Calls `smooth::auto.gum()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.auto_gum")
lrn("fcst.auto_gum")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels	Range
orders	integer	3		$[1, \infty)$
lags	integer	-		$[1, \infty)$
type	character	additive	additive, multiplicative, select	-
initial	character	backcasting	backcasting, optimal, two-stage, complete	-
ic	character	AICc	AICc, AIC, BIC, BICc	-
loss	character	likelihood	likelihood, MSE, MAE, HAM, MSEh, TMSE, GTMSE, MSCE, GPL	-
holdout	logical	FALSE	TRUE, FALSE	-
bounds	character	usual	usual, admissible, none	-
silent	logical	TRUE	TRUE, FALSE	-
regressors	character	use	use, select, adapt, integrate	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstAutoGum`

Methods

Public methods:

- `LearnerFcstAutoGum$new()`
- `LearnerFcstAutoGum$clone()`

`LearnerFcstAutoGum$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstAutoGum$new()
```

`LearnerFcstAutoGum$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstAutoGum$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`,

mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarir mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.auto_gum")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.auto_msarima

Auto Multiple-Seasonal ARIMA Forecast Learner

Description

Automatic order selection for Multiple-Seasonal ARIMA. Picks `orders` minimizing the chosen information criterion. Calls `smooth::auto.msarima()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.auto_msarima")
lrn("fcst.auto_msarima")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels
orders	untyped	list(ar = c(3, 3), i = c(2, 1), ma = c(3, 3))	
lags	untyped	-	
initial	character	backcasting	backcasting, optimal, two-stage, complete
ic	character	AICc	AICc, AIC, BIC, BICc
loss	character	likelihood	likelihood, MSE, MAE, HAM, MSEh, TMSE, GTMSE, M
holdout	logical	FALSE	TRUE, FALSE
bounds	character	usual	usual, admissible, none
silent	logical	TRUE	TRUE, FALSE
regressors	character	use	use, select, adapt

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstAutoMsarima`

Methods

Public methods:

- `LearnerFcstAutoMsarima$new()`
- `LearnerFcstAutoMsarima$clone()`

`LearnerFcstAutoMsarima$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstAutoMsarima$new()
```

`LearnerFcstAutoMsarima$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstAutoMsarima$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

Svetunkov I (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)*, 1st edition. Chapman and Hall/CRC. doi:10.1201/9781003452652. https://openforecast.org/adam/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarir mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.auto_msarima")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)
```

```
# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.auto_ssarima`
Auto State-Space ARIMA Forecast Learner

Description

Automatic order selection for State-Space ARIMA. Picks `orders` minimizing the chosen information criterion. Calls `smooth::auto.ssarima()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.auto_ssarima")
lrn("fcst.auto_ssarima")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels
orders	untyped	list(ar = c(3, 3), i = c(2, 1), ma = c(3, 3))	
lags	untyped	-	
fast	logical	TRUE	TRUE, FALSE
constant	logical	NULL	TRUE, FALSE

initial	character	backcasting	backcasting, optimal, two-stage, complete
ic	character	AICc	AICc, AIC, BIC, BICc
loss	character	likelihood	likelihood, MSE, MAE, HAM, MSEh, TMSE, GTMSE, N
holdout	logical	FALSE	TRUE, FALSE
bounds	character	admissible	admissible, usual, none
silent	logical	TRUE	TRUE, FALSE
regressors	character	use	use, select, adapt

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstAutoSsarima`

Methods

Public methods:

- `LearnerFcstAutoSsarima$new()`
- `LearnerFcstAutoSsarima$clone()`

`LearnerFcstAutoSsarima$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstAutoSsarima$new()
```

`LearnerFcstAutoSsarima$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstAutoSsarima$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

Svetunkov I (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)*, 1st edition. Chapman and Hall/CRC. doi:10.1201/9781003452652. https://openforecast.org/adam/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners

- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarir` `mlr_learners_fcst.nnetar`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.random_walk`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.stlm`, `mlr_learners_fcst.struct_ts`, `mlr_learners_fcst.tbats` `mlr_learners_fcst.theta`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.auto_ssarima")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.bagged`

Bagged Model Forecast Learner

Description

Bootstrap-aggregated forecasts. The series is resampled via the Box-Cox/Loess moving block bootstrap of Bergmeir, Hyndman, and Benítez and `fn` is fit on each replicate. The forecast averages across the ensemble. Calls `forecast::baggedModel()` from package **forecast**.

`fn` is the model-fitting function applied to each bootstrap replicate (defaults to `forecast::ets()`). Any function returning an object compatible with `forecast::forecast()` may be passed, e.g. `forecast::auto.arima()` or `forecast::Arima()` with fixed orders via a wrapper. The number of bootstrap replicates is controlled by `num`, and `block_size` configures the moving block length in `forecast::bld.mbb.bootstrap()`. Prediction intervals from `forecast::forecast.baggedModel()` are the empirical bootstrap range (not configurable by `level`), so only `"response"` is offered as a `predict_type`.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.bagged")
lrn("fcst.bagged")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Range
fn	untyped	-	-
num	integer	100	$[1, \infty)$
block_size	integer	NULL	$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstBaggedModel`

Methods

Public methods:

- `LearnerFcstBaggedModel$new()`
- `LearnerFcstBaggedModel$clone()`

`LearnerFcstBaggedModel$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstBaggedModel$new()
```

`LearnerFcstBaggedModel$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstBaggedModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Bergmeir C, Hyndman RJ, Benítez JM (2016). "Bagging exponential smoothing methods using STL decomposition and Box-Cox transformation." *International Journal of Forecasting*, **32**(2), 303–312. doi:10.1016/j.ijforecast.2015.07.002.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croste`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarim`

mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats, mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.bagged")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.bats

BATS Forecast Learner

Description

Exponential smoothing state space model with Box-Cox transformation, ARMA errors, Trend and Seasonal components (BATS) model. Calls forecast::bats() from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function mlr3::lrn():

```
mlr_learners$get("fcst.bats")
lrn("fcst.bats")
```

Meta Information

- Task type: “fcst”
- Predict Types: “response”, “quantiles”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”, “POSIXct”, “Date”
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
use.box.cox	logical	NULL	TRUE, FALSE	-
use.trend	logical	NULL	TRUE, FALSE	-
use.damped.trend	logical	NULL	TRUE, FALSE	-
seasonal.periods	untyped	NULL		-
use.arma.errors	logical	TRUE	TRUE, FALSE	-
use.parallel	logical	-	TRUE, FALSE	-
num.cores	integer	2		$[1, \infty)$
bc.lower	numeric	0		$(-\infty, \infty)$
bc.upper	numeric	1		$(-\infty, \infty)$
biasadj	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstBats`

Methods

Public methods:

- `LearnerFcstBats$new()`
- `LearnerFcstBats$clone()`

`LearnerFcstBats$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstBats$new()
```

`LearnerFcstBats$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstBats$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

De Livera AM, Hyndman RJ, Snyder RD (2011). "Forecasting time series with complex seasonal patterns using exponential smoothing." *Journal of the American Statistical Association*, **106**(496), 1513–1527.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.ces, mlr_learners_fcst.cros mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarir mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.bats")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.ces` *CES Forecast Learner*

Description

Complex Exponential Smoothing (CES) model. Calls `smooth::ces()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.ces")
lrn("fcst.ces")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels
seasonality	character	none	none, simple, partial, full
lags	untyped	-	
initial	character	backcasting	backcasting, optimal, two-stage, complete
a	untyped	NULL	
b	untyped	NULL	
loss	character	likelihood	likelihood, MSE, MAE, HAM, MSEh, TMSE, GTMSE, MSCE, GPL
holdout	logical	FALSE	TRUE, FALSE
bounds	character	admissible	admissible, none

silent	logical	TRUE	TRUE, FALSE
regressors	character	use	use, select, adapt

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstCes`

Methods

Public methods:

- `LearnerFcstCes$new()`
- `LearnerFcstCes$clone()`

`LearnerFcstCes$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstCes$new()
```

`LearnerFcstCes$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstCes$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

Svetunkov I (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)*, 1st edition. Chapman and Hall/CRC. doi:10.1201/9781003452652. https://openforecast.org/adam/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.

- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.rando mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.ces")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.croston`

Croston Forecast Learner

Description

Croston model. Calls `forecast::croston_model()` from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.croston")
lrn("fcst.croston")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
alpha	numeric	0.1		$[0, 1]$
type	character	croston	croston, sba, sbj	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstCroston`

Methods

Public methods:

- `LearnerFcstCroston$new()`
- `LearnerFcstCroston$clone()`

`LearnerFcstCroston$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstCroston$new()
```

`LearnerFcstCroston$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstCroston$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Croston JD (1972). "Forecasting and stock control for intermittent demands." *Journal of the Operational Research Society*, **23**(3), 289–303.

Shale EA, Boylan JE, Johnston F (2006). "Forecasting for intermittent demand: the estimation of an unbiased average." *Journal of the Operational Research Society*, **57**(5), 588–592.

Shenstone L, Hyndman RJ (2005). "Stochastic models underlying Croston's method for intermittent demand forecasting." *Journal of Forecasting*, **24**(6), 389–402.

Syntetos AA, Boylan JE (2001). "On the bias of intermittent demand estimates." *International Journal of Production Economics*, **71**(1-3), 457–466.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.rando mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.croston")
print(learner)

# Define a Task
task = tsk("airpassengers")
```

```
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.elm` *Extreme Learning Machine Forecast Learner*

Description

Automatic time series forecasting with an extreme learning machine neural network, including automatic input lag selection, deterministic seasonality handling, and ensemble combination across multiple training repetitions. Calls `nnfor::elm()` from package **nnfor**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.elm")
lrn("fcst.elm")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date"
- Required Packages: **mlr3**, **mlr3forecast**, **nnfor**, **forecast**

Parameters

Id	Type	Default	Levels	Range
m	integer	-		$[1, \infty)$
hd	untyped	NULL		-
type	character	lasso	lasso, ridge, step, lm	-
reps	integer	20		$[1, \infty)$
comb	character	median	median, mean, mode	-
lags	untyped	NULL		-
keep	untyped	NULL		-
differorder	untyped	NULL		-
sel.lag	logical	TRUE	TRUE, FALSE	-
direct	logical	FALSE	TRUE, FALSE	-
allow.det.season	logical	TRUE	TRUE, FALSE	-
det.type	character	auto	auto, bin, trg	-
barebone	logical	FALSE	TRUE, FALSE	-
retrain	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstElm`

Methods

Public methods:

- `LearnerFcstElm$new()`
- `LearnerFcstElm$clone()`

`LearnerFcstElm$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstElm$new()
```

`LearnerFcstElm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstElm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Kourentzes N, Barrow DK, Crone SF (2014). "Neural network ensemble operators for time series forecasting." *Expert Systems with Applications*, **41**(9), 4235–4244. doi:10.1016/j.eswa.2013.12.011.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.rando mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.elm")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.es` *Exponential Smoothing Forecast Learner*

Description

Exponential smoothing (ETS) model in state-space form. Supports multiple seasonal lags natively (e.g. `lags = c(1, 24, 168)` for hourly data with daily and weekly cycles). Calls `smooth::es()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.es")
lrn("fcst.es")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels	Range
model	untyped	"ZXZ"		-
lags	untyped	-		-
persistence	untyped	NULL		-
phi	numeric	NULL		$(-\infty, \infty)$
initial	character	backcasting	backcasting, optimal, two-stage, complete	-
initialSeason	untyped	NULL		-
ic	character	AICc	AICc, AIC, BIC, BICc	-
loss	character	likelihood	likelihood, MSE, MAE, HAM, MSEh, TMSE, GTMSE, MSCE, GPL	-
holdout	logical	FALSE	TRUE, FALSE	-
bounds	character	usual	usual, admissible, none	-
silent	logical	TRUE	TRUE, FALSE	-
regressors	character	use	use, select	-

Super classes

mlr3::Learner -> mlr3::LearnerRegr -> LearnerFcst -> LearnerFcstSmooth -> LearnerFcstEs

Methods

Public methods:

- LearnerFcstEs$new()
- LearnerFcstEs$clone()

`LearnerFcstEs$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstEs$new()
```

`LearnerFcstEs$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstEs$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

Svetunkov I (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)*, 1st edition. Chapman and Hall/CRC. doi:10.1201/9781003452652. https://openforecast.org/adam/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:

  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.rando mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.es")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.ets *ETS Forecast Learner*

Description

Exponential Smoothing State Space (ETS) model. Calls forecast::ets() from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.ets")
lrn("fcst.ets")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
model	untyped	"ZZZ"		-
damped	logical	NULL	TRUE, FALSE	-
alpha	numeric	NULL		$(-\infty, \infty)$
beta	numeric	NULL		$(-\infty, \infty)$
gamma	numeric	NULL		$(-\infty, \infty)$
phi	numeric	NULL		$(-\infty, \infty)$
additive.only	logical	FALSE	TRUE, FALSE	-
lambda	untyped	NULL		-
biasadj	logical	FALSE	TRUE, FALSE	-
lower	untyped	c(rep.int(1e-04, 3), 0.8)		-
upper	untyped	c(rep.int(0.9999, 3), 0.98)		-
opt.crit	character	lik	lik, amse, mse, sigma, mae	-
nmse	integer	3		$[1, 30]$
bounds	character	both	both, usual, admissible	-
ic	character	aicc	aicc, aic, bic	-
restrict	logical	TRUE	TRUE, FALSE	-
allow.multiplicative.trend	logical	FALSE	TRUE, FALSE	-
simulate	logical	FALSE	TRUE, FALSE	-
bootstrap	logical	FALSE	TRUE, FALSE	-
npaths	integer	5000		$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstEts`

Methods

Public methods:

- `LearnerFcstEts$new()`
- `LearnerFcstEts$clone()`

`LearnerFcstEts$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstEts$new()
```

`LearnerFcstEts$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstEts$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Hyndman RJ, Koehler AB, Snyder RD, Grose S (2002). "A state space framework for automatic forecasting using exponential smoothing methods." *International Journal of Forecasting*, **18**(3), 439–454.

Hyndman RJ, Akram M, Archibald B (2008). "The admissible parameter space for exponential smoothing models." *Annals of the Institute of Statistical Mathematics*, **60**(2), 407–426.

Hyndman RJ, Koehler AB, Ord JK, Snyder RD (2008). *Forecasting with exponential smoothing: the state space approach*. Springer-Verlag. https://robjhyndman.com/expsmooth/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`,

mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.rando mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.ets")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.gum *GUM Forecast Learner*

Description

Generalised Univariate Model (GUM): a single-source-of-error state-space model with a user-defined transition matrix, persistence vector and measurement vector. Generalises exponential smoothing beyond the ETS structural template. Calls `smooth::gum()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.gum")
lrn("fcst.gum")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels
orders	untyped	c(1, 1)	
lags	untyped	-	
type	character	additive	additive, multiplicative
initial	character	backcasting	backcasting, optimal, two-stage, complete
persistence	untyped	NULL	
transition	untyped	NULL	
measurement	untyped	-	
loss	character	likelihood	likelihood, MSE, MAE, HAM, MSEh, TMSE, GTMSE, MSCE, GPL
holdout	logical	FALSE	TRUE, FALSE
bounds	character	usual	usual, admissible, none
silent	logical	TRUE	TRUE, FALSE
regressors	character	use	use, select, adapt, integrate

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstGum`

Methods

Public methods:

- `LearnerFcstGum$new()`
- `LearnerFcstGum$clone()`

`LearnerFcstGum$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstGum$new()
```

`LearnerFcstGum$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstGum$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.rando mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.gum")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.holt_winters`
Holt-Winters Forecast Learner

Description

Holt-Winters exponential smoothing with optional trend and additive or multiplicative seasonal component. Smoothing parameters are estimated by minimizing the squared one-step prediction error. Calls `stats::HoltWinters()` from package **stats** and forecasts via `forecast::forecast()`.

Setting `beta = FALSE` fits a simple exponential smoothing model (no trend). Setting `gamma = FALSE` fits a non-seasonal model.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.holt_winters")
lrn("fcst.holt_winters")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
alpha	numeric	NULL		$[0, 1]$
beta	numeric	NULL		$[0, 1]$

gamma	numeric	NULL		$[0, 1]$
seasonal	character	additive	additive, multiplicative	-
start.periods	integer	2		$[2, \infty)$
l.start	numeric	NULL		$(-\infty, \infty)$
b.start	numeric	NULL		$(-\infty, \infty)$
s.start	untyped	NULL		-
optim.start	untyped	c(alpha = 0.3, beta = 0.1, gamma = 0.1)		-
optim.control	untyped	list()		-
lambda	untyped	NULL		-
biasadj	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstHoltWinters`

Methods

Public methods:

- `LearnerFcstHoltWinters$new()`
- `LearnerFcstHoltWinters$clone()`

`LearnerFcstHoltWinters$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstHoltWinters$new()
```

`LearnerFcstHoltWinters$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstHoltWinters$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Holt CC (2004). "Forecasting seasonals and trends by exponentially weighted moving averages." *International Journal of Forecasting*, **20**(1), 5–10. doi:10.1016/j.ijforecast.2003.09.015.

Winters PR (1960). "Forecasting Sales by Exponentially Weighted Moving Averages." *Management Science*, **6**(3), 324–342. doi:10.1287/mnsc.6.3.324.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package **mlr3learners** for a solid collection of essential learners.

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarima`, `mlr_learners_fcst.nnetar`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.rando`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.stlm`, `mlr_learners_fcst.struct_ts`, `mlr_learners_fcst.tbats`, `mlr_learners_fcst.theta`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.holt_winters")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.mean` *Mean Forecast Learner*

Description

Mean model. Calls `forecast::mean_model()` from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.mean")
lrn("fcst.mean")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
lambda	untyped	NULL		-
biasadj	logical	FALSE	TRUE, FALSE	-
bootstrap	logical	FALSE	TRUE, FALSE	-
npaths	integer	5000		$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstMean`

Methods

Public methods:

- `LearnerFcstMean$new()`
- `LearnerFcstMean$clone()`

`LearnerFcstMean$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstMean$new()
```

`LearnerFcstMean$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstMean$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Hyndman RJ, Athanasopoulos G (2018). *Forecasting: principles and practice*, 2nd edition. OTexts, Melbourne, Australia. https://OTexts.com/fpp2/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarima`, `mlr_learners_fcst.nnetar`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.rando`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.stlm`, `mlr_learners_fcst.struct_ts`, `mlr_learners_fcst.tbats`, `mlr_learners_fcst.theta`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.mean")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.mlp *Multilayer Perceptron Forecast Learner*

Description

Automatic time series forecasting with a multilayer perceptron neural network, including automatic input lag selection, deterministic seasonality handling, and ensemble combination across multiple training repetitions. Calls `nnfor::mlp()` from package **nnfor**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.mlp")
lrn("fcst.mlp")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response"

- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”, “POSIXct”, “Date”
- Required Packages: **mlr3**, **mlr3forecast**, **nnfor**, **forecast**

Parameters

Id	Type	Default	Levels	Range
m	integer	-		$[1, \infty)$
hd	untyped	NULL		-
reps	integer	20		$[1, \infty)$
comb	character	median	median, mean, mode	-
lags	untyped	NULL		-
keep	untyped	NULL		-
difforder	untyped	NULL		-
sel.lag	logical	TRUE	TRUE, FALSE	-
allow.det.season	logical	TRUE	TRUE, FALSE	-
det.type	character	auto	auto, bin, trg	-
hd.auto.type	character	set	set, valid, cv, elm	-
hd.max	integer	NULL		$[1, \infty)$
retrain	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstMlp`

Methods

Public methods:

- `LearnerFcstMlp$new()`
- `LearnerFcstMlp$clone()`

`LearnerFcstMlp$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstMlp$new()
```

`LearnerFcstMlp$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstMlp$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Kourentzes N, Barrow DK, Crone SF (2014). “Neural network ensemble operators for time series forecasting.” *Expert Systems with Applications*, **41**(9), 4235–4244. doi:10.1016/j.eswa.2013.12.011.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.msarima`, `mlr_learners_fcst.nnetar`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.rando`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.stlm`, `mlr_learners_fcst.struct_ts`, `mlr_learners_fcst.tbats`, `mlr_learners_fcst.theta`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.mlp")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.msarima`

Multiple-Seasonal ARIMA Forecast Learner

Description

Multiple-Seasonal ARIMA model in state-space form. Supports multiple seasonal lags natively (e.g. `lags = c(1, 24, 168)` for hourly data with daily and weekly cycles). Calls `smooth::msarima()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.msarima")
lrn("fcst.msarima")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels
orders	untyped	list(ar = 0, i = 1, ma = 1)	
lags	untyped	1	
constant	logical	FALSE	TRUE, FALSE
arma	untyped	NULL	
initial	character	backcasting	backcasting, optimal, two-stage, complete
ic	character	AICc	AICc, AIC, BIC, BICc
loss	character	likelihood	likelihood, MSE, MAE, HAM, MSEh, TMSE, GTMSE, MSCE, GPL
holdout	logical	FALSE	TRUE, FALSE
bounds	character	usual	usual, admissible, none

silent	logical	TRUE	TRUE, FALSE
regressors	character	use	use, select, adapt

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstMsarima`

Methods

Public methods:

- `LearnerFcstMsarima$new()`
- `LearnerFcstMsarima$clone()`

`LearnerFcstMsarima$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstMsarima$new()
```

`LearnerFcstMsarima$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstMsarima$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

Svetunkov I (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)*, 1st edition. Chapman and Hall/CRC. doi:10.1201/9781003452652. https://openforecast.org/adam/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.

- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_wa mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.msarima")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.nnetar

Neural Network Forecast Learner

Description

Single Layer Neural Network. Calls `forecast::nnetar()` from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.nnetar")
lrn("fcst.nnetar")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**, **nnet**

Parameters

Id	Type	Default	Levels	Range
p	integer	-		$[0, \infty)$
P	integer	1		$[0, \infty)$
size	integer	NULL		$[1, \infty)$
repeats	integer	20		$(-\infty, \infty)$
lambda	untyped	NULL		-
scale.inputs	logical	TRUE	TRUE, FALSE	-
parallel	logical	FALSE	TRUE, FALSE	-
num.cores	integer	2		$[1, \infty)$
bootstrap	logical	FALSE	TRUE, FALSE	-
npaths	integer	1000		$[1, \infty)$
innov	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstNnetar`

Methods

Public methods:

- `LearnerFcstNnetar$new()`
- `LearnerFcstNnetar$clone()`

`LearnerFcstNnetar$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstNnetar$new()
```

`LearnerFcstNnetar$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstNnetar$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Ripley BD (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press. doi:10.1017/cbo9780511812651.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarima`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.random_w`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.stlm`, `mlr_learners_fcst.struct_ts`, `mlr_learners_fcst.tbats`, `mlr_learners_fcst.theta`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.nnetar")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.prophet

Prophet Forecast Learner

Description

Prophet model. Calls `prophet::prophet()` from package **prophet**.

With `growth = "logistic"`, the task must have a feature named `cap` containing the saturation capacity. An optional `floor` feature specifies the saturation minimum. These columns must also be supplied for future prediction rows and are not registered as extra regressors.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.prophet")
lrn("fcst.prophet")
```

Meta Information

- Task type: “fcst”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3forecast**, **prophet**

Parameters

Id	Type	Default	Levels	Range
growth	character	linear	linear, logistic, flat	-
changepoints	untyped	NULL		-
n.changepoints	integer	25		$[0, \infty)$
changepoint.range	numeric	0.8		$[0, 1]$
yearly.seasonality	untyped	"auto"		-
weekly.seasonality	untyped	"auto"		-
daily.seasonality	untyped	"auto"		-
holidays	untyped	NULL		-
seasonality.mode	character	additive	additive, multiplicative	-
seasonality.prior.scale	numeric	10		$[0, \infty)$
holidays.prior.scale	numeric	10		$[0, \infty)$
changepoint.prior.scale	numeric	0.05		$[0, \infty)$
mcmc.samples	integer	0		$[0, \infty)$
interval.width	numeric	0.8		$[0, 1]$
uncertainty.samples	integer	1000		$[0, \infty)$
backend	character	NULL	rstan, cmdstanr	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstProphet`

Methods

Public methods:

- `LearnerFcstProphet$new()`
- `LearnerFcstProphet$clone()`

`LearnerFcstProphet$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstProphet$new()
```

`LearnerFcstProphet$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstProphet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Taylor SJ, Letham B (2018). "Forecasting at Scale." *The American Statistician*, **72**(1), 37–45. doi:10.1080/00031305.2017.1380080.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.random_wa mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.prophet")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.random_walk`
Random Walk Forecast Learner

Description

Random walk model. Calls `forecast::rw_model()` from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.random_walk")
lrn("fcst.random_walk")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
lag	integer	1		$[1, \infty)$
drift	logical	FALSE	TRUE, FALSE	-
lambda	untyped	NULL		-
biasadj	logical	FALSE	TRUE, FALSE	-
simulate	logical	FALSE	TRUE, FALSE	-
bootstrap	logical	FALSE	TRUE, FALSE	-
npaths	integer	5000		$[1, \infty)$

Super classes

mlr3::Learner -> mlr3::LearnerRegr -> LearnerFcst -> LearnerFcstForecast -> LearnerFcstRandomWalk

Methods

Public methods:

- LearnerFcstRandomWalk$new()
- LearnerFcstRandomWalk$clone()

LearnerFcstRandomWalk$new(): Creates a new instance of this R6 class.

Usage:

```
LearnerFcstRandomWalk$new()
```

LearnerFcstRandomWalk$clone(): The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstRandomWalk$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats,

mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats, mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.random_walk")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.rlgt

Local and Global Trend Forecast Learner

Description

Bayesian exponential smoothing with a nonlinear global trend (LGT/SGT), Student-t errors, and optional heteroscedasticity, fitted via MCMC. The seasonal period is taken from the frequency of the series. Calls `Rlgt::rlgt()` from package **Rlgt**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.rlgt")
lrn("fcst.rlgt")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **Rlgt**

Parameters

Id	Type	Default	Levels	Range
seasonality	integer	1		$[1, \infty)$
seasonality2	integer	1		$[1, \infty)$
seasonality.type	character	multiplicative	multiplicative, generalized	-
error.size.method	character	std	std, innov	-
level.method	character	HW	HW, seasAvg, HW_sAvg	-
method	character	Gibbs	Gibbs, Stan	-
homoscedastic	logical	FALSE	TRUE, FALSE	-
control	untyped	-		-
verbose	logical	FALSE	TRUE, FALSE	-
NUM_OF_TRIALS	integer	2000		$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstRlgt`

Methods

Public methods:

- `LearnerFcstRlgt$new()`
- `LearnerFcstRlgt$clone()`

`LearnerFcstRlgt$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstRlgt$new()
```

`LearnerFcstRlgt$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstRlgt$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Smyl S, Bergmeir C, Wibowo E, Ng TW, Long X, Dokumentov A, Schmidt D (2025). *Rlgt: Bayesian Exponential Smoothing Models with Trend Modifications*. R package version 0.2-3, https://github.com/cbergmeir/Rlgt.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.splin mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.rlgt")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
```

```
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.sma` *Simple Moving Average Forecast Learner*

Description

Simple moving average. The forecast is the mean of the last `order` observations. If `order` is `NULL` (the default), the optimal window is selected automatically according to the chosen information criterion `ic`. Calls `smooth::sma()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.sma")
lrn("fcst.sma")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels	Range
order	integer	NULL		$[1, \infty)$
ic	character	AICc	AICc, AIC, BIC, BICc	-
silent	logical	TRUE	TRUE, FALSE	-

fast	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSma`

Methods

Public methods:

- `LearnerFcstSma$new()`
- `LearnerFcstSma$clone()`

`LearnerFcstSma$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstSma$new()
```

`LearnerFcstSma$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstSma$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sparma, mlr_learners_fcst.spli mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.sma")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.sparma

Sparse ARMA Forecast Learner

Description

Sparse ARMA model in state-space form. Unlike ARIMA, which expands polynomials, the AR and MA orders map to specific lags, so `orders = list(ar = c(1, 12), ma = 0)` fits terms at lags 1 and 12 and nothing in between. Calls `smooth::sparma()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.sparma")
lrn("fcst.sparma")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels
orders	untyped	list(ar = 1, ma = 1)	
constant	logical	FALSE	TRUE, FALSE
arma	untyped	NULL	
initial	character	backcasting	backcasting, optimal, two-stage, complete
loss	character	likelihood	likelihood, MSE, MAE, HAM, LASSO, RIDGE, MSEh, TMSE, GTMSE, MSCE
holdout	logical	FALSE	TRUE, FALSE
bounds	character	none	none, usual, admissible
silent	logical	TRUE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstSparma`

Methods

Public methods:

- `LearnerFcstSparma$new()`
- `LearnerFcstSparma$clone()`

`LearnerFcstSparma$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstSparma$new()
```

`LearnerFcstSparma$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstSparma$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

Svetunkov I (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)*, 1st edition. Chapman and Hall/CRC. doi:10.1201/9781003452652. https://openforecast.org/adam/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats, mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.sparma")
print(learner)
```

```
# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.spline`

Spline Forecast Learner

Description

Cubic spline stochastic model. Calls `forecast::spline_model()` from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.spline")
lrn("fcst.spline")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
method	character	gcv	gcv, mle	-
lambda	untyped	NULL		-
biasadj	logical	FALSE	TRUE, FALSE	-
simulate	logical	FALSE	TRUE, FALSE	-
bootstrap	logical	FALSE	TRUE, FALSE	-
npaths	integer	5000		$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstSpline`

Methods

Public methods:

- `LearnerFcstSpline$new()`
- `LearnerFcstSpline$clone()`

`LearnerFcstSpline$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstSpline$new()
```

`LearnerFcstSpline$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstSpline$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Hyndman RJ, King ML, Pitrun I, Billah B (2005). "Local linear forecasts using cubic smoothing splines." *Australian & New Zealand Journal of Statistics*, **47**(1), 87–99.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.

- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats, mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.spline")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.ssarima`

State-Space ARIMA Forecast Learner

Description

State-Space ARIMA model. Supports multiple seasonal lags natively (e.g. `lags = c(1, 24, 168)` for hourly data with daily and weekly cycles). Calls `smooth::ssarima()` from package **smooth**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.ssarima")
lrn("fcst.ssarima")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **smooth**

Parameters

Id	Type	Default	Levels
orders	untyped	list(ar = 0, i = 1, ma = 1)	
lags	untyped	-	
constant	logical	FALSE	TRUE, FALSE
arma	untyped	NULL	
initial	character	backcasting	backcasting, optimal, two-stage, complete
loss	character	likelihood	likelihood, MSE, MAE, HAM, MSEh, TMSE, GTMSE, MSCE, GPL
holdout	logical	FALSE	TRUE, FALSE
bounds	character	admissible	admissible, usual, none
silent	logical	TRUE	TRUE, FALSE
regressors	character	use	use, select, adapt

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstSmooth` -> `LearnerFcstSsarima`

Methods

Public methods:

- `LearnerFcstSsarima$new()`
- `LearnerFcstSsarima$clone()`

`LearnerFcstSsarima$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstSsarima$new()
```

`LearnerFcstSsarima$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstSsarima$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Svetunkov I (2023). "Smooth forecasting with the smooth package in R." 2301.01790, https://arxiv.org/abs/2301.01790.

Svetunkov I (2023). *Forecasting and Analytics with the Augmented Dynamic Adaptive Model (ADAM)*, 1st edition. Chapman and Hall/CRC. doi:10.1201/9781003452652. https://openforecast.org/adam/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`,

mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_ts, mlr_learners_fcst.tbats, mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.ssarima")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.stlm

STL + ETS/ARIMA Forecast Learner

Description

Forecasts of seasonal time series using STL decomposition. The seasonal component is forecast naively and the seasonally-adjusted series is forecast with either an ETS or ARIMA model. Calls forecast::stlm() from package **forecast**.

The task must provide a seasonal time series (frequency > 1).

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.stlm")
lrn("fcst.stlm")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
s.window	untyped	7L + 4L * seq_len(6L)		-
t.window	integer	NULL		$[1, \infty)$
robust	logical	FALSE	TRUE, FALSE	-
method	character	ets	ets, arima	-
modelfunction	untyped	NULL		-
etsmodel	untyped	"ZZN"		-
lambda	untyped	NULL		-
biasadj	logical	FALSE	TRUE, FALSE	-
allow.multiplicative.trend	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstStlm`

Methods

Public methods:

- `LearnerFcstStlm$new()`
- `LearnerFcstStlm$clone()`

`LearnerFcstStlm$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstStlm$new()
```

`LearnerFcstStlm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstStlm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Cleveland RB, Cleveland WS, McRae JE, Terpenning I (1990). "STL: A Seasonal-Trend Decomposition Procedure Based on Loess." *Journal of Official Statistics*, **6**(1), 3–73.

Hyndman RJ, Athanasopoulos G (2018). *Forecasting: principles and practice*, 2nd edition. OTexts, Melbourne, Australia. https://OTexts.com/fpp2/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarima`, `mlr_learners_fcst.nnetar`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.random_walk`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.struct_ts`, `mlr_learners_fcst.tbats`, `mlr_learners_fcst.theta`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.stlm")
print(learner)

# Define a Task
```

```
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.struct_ts`

Structural Time Series Forecast Learner

Description

Structural time series model fit by maximum likelihood. Three model types are supported: local level, local linear trend, and basic structural model (level + trend + seasonal). Calls `stats::StructTS()` from package **stats**.

`type = "BSM"` requires a seasonal time series (frequency > 1). Prediction is performed via `forecast::forecast.StructTS()` which yields point forecasts and predictive intervals from the Kalman filter.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.struct_ts")
lrn("fcst.struct_ts")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels
type	character	level	level, trend, BSM
init	untyped	NULL	
fixed	untyped	NULL	
optim.control	untyped	NULL	
lambda	untyped	NULL	
biasadj	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstStructTS`

Methods

Public methods:

- `LearnerFcstStructTS$new()`
- `LearnerFcstStructTS$clone()`

`LearnerFcstStructTS$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstStructTS$new()
```

`LearnerFcstStructTS$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstStructTS$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Harvey AC (1989). *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press, Cambridge.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners

- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarima`, `mlr_learners_fcst.nnetar`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.random_walk`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.stlm`, `mlr_learners_fcst.tbats`, `mlr_learners_fcst.theta`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.struct_ts")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.tbats` *TBATS Forecast Learner*

Description

Exponential smoothing state space model with Box-Cox transformation, ARMA errors, Trend and Seasonal components (TBATS) model. Calls `forecast::tbats()` from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.tbats")
lrn("fcst.tbats")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct", "Date"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels	Range
use.box.cox	logical	NULL	TRUE, FALSE	-
use.trend	logical	NULL	TRUE, FALSE	-
use.damped.trend	logical	NULL	TRUE, FALSE	-
seasonal.periods	untyped	NULL		-
use.arma.errors	logical	TRUE	TRUE, FALSE	-
use.parallel	logical	-	TRUE, FALSE	-
num.cores	integer	2		$[1, \infty)$
bc.lower	numeric	0		$(-\infty, \infty)$
bc.upper	numeric	1		$(-\infty, \infty)$
biasadj	logical	FALSE	TRUE, FALSE	-
simulate	logical	FALSE	TRUE, FALSE	-
bootstrap	logical	FALSE	TRUE, FALSE	-
npaths	integer	5000		$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstTbats`

Methods

Public methods:

- `LearnerFcstTbats$new()`
- `LearnerFcstTbats$clone()`

`LearnerFcstTbats$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstTbats$new()
```

`LearnerFcstTbats$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstTbats$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

De Livera AM, Hyndman RJ, Snyder RD (2011). "Forecasting time series with complex seasonal patterns using exponential smoothing." *Journal of the American Statistical Association*, **106**(496), 1513–1527.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`,

mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_t mlr_learners_fcst.theta, mlr_learners_fcst.tscount, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.tbats")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.theta

Theta Forecast Learner

Description

Theta model. Calls `forecast::theta_model()` from package **forecast**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.theta")
lrn("fcst.theta")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels
lambda	untyped	NULL	
biasadj	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstTheta`

Methods

Public methods:

- `LearnerFcstTheta$new()`
- `LearnerFcstTheta$clone()`

`LearnerFcstTheta$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstTheta$new()
```

`LearnerFcstTheta$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstTheta$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Assimakopoulos V, Nikolopoulos K (2000). "The theta model: a decomposition approach to forecasting." *International Journal of Forecasting*, **16**(4), 521–530.

Hyndman RJ, Billah B (2003). "Unmasking the Theta method." *International Journal of Forecasting*, **19**(2), 287–290.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`, `mlr_learners_fcst.mlp`, `mlr_learners_fcst.msarima`, `mlr_learners_fcst.nnetar`, `mlr_learners_fcst.prophet`, `mlr_learners_fcst.random_walk`, `mlr_learners_fcst.rlgt`, `mlr_learners_fcst.sma`, `mlr_learners_fcst.sparma`, `mlr_learners_fcst.spline`, `mlr_learners_fcst.ssarima`, `mlr_learners_fcst.stlm`, `mlr_learners_fcst.struct_t`, `mlr_learners_fcst.tbats`, `mlr_learners_fcst.tscount`, `mlr_learners_fcst.tslm`

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.theta")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_fcst.tscount`

Count Time Series Forecast Learner

Description

Generalized linear model for count time series (INGARCH). Calls `tscount::tsglm()` from package **tscount**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("fcst.tscount")
lrn("fcst.tscount")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **tscount**

Parameters

Id	Type	Default	Levels	Range
past_obs	untyped	NULL		-
past_mean	untyped	NULL		-
external	untyped	FALSE		-
link	character	identity	identity, log	-
distr	character	poisson	poisson, nbinom	-
B	integer	1000		$[10, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstTscount`

Methods

Public methods:

- `LearnerFcstTscount$new()`
- `LearnerFcstTscount$clone()`

`LearnerFcstTscount$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstTscount$new()
```

`LearnerFcstTscount$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstTscount$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

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.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `LearnerFcst`, `mlr_learners_fcst.adam`, `mlr_learners_fcst.ar`, `mlr_learners_fcst.arfima`, `mlr_learners_fcst.arima`, `mlr_learners_fcst.auto_adam`, `mlr_learners_fcst.auto_arima`, `mlr_learners_fcst.auto_ces`, `mlr_learners_fcst.auto_gum`, `mlr_learners_fcst.auto_msarima`, `mlr_learners_fcst.auto_ssarima`, `mlr_learners_fcst.bagged`, `mlr_learners_fcst.bats`, `mlr_learners_fcst.ces`, `mlr_learners_fcst.croston`, `mlr_learners_fcst.elm`, `mlr_learners_fcst.es`, `mlr_learners_fcst.ets`, `mlr_learners_fcst.gum`, `mlr_learners_fcst.holt_winters`, `mlr_learners_fcst.mean`,

mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_t mlr_learners_fcst.tbats, mlr_learners_fcst.theta, mlr_learners_fcst.tslm

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.tscount")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_fcst.tslm

Time Series Linear Model Forecast Learner

Description

Time series linear model. Calls forecast::tslm() from package **forecast**.

If formula is not set, the model is fit with the trend term of forecast::tslm() plus all features. The season term is included when the task frequency is greater than one.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function mlr3::lrn():

```
mlr_learners$get("fcst.tslm")
lrn("fcst.tslm")
```

Meta Information

- Task type: "fcst"
- Predict Types: "response", "quantiles"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3forecast**, **forecast**

Parameters

Id	Type	Default	Levels
formula	untyped	-	
lambda	untyped	NULL	
biasadj	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerFcst` -> `LearnerFcstForecast` -> `LearnerFcstTslm`

Methods

Public methods:

- `LearnerFcstTslm$new()`
- `LearnerFcstTslm$clone()`

`LearnerFcstTslm$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerFcstTslm$new()
```

`LearnerFcstTslm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerFcstTslm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Hyndman RJ, Athanasopoulos G (2018). *Forecasting: principles and practice*, 2nd edition. OTexts, Melbourne, Australia. https://OTexts.com/fpp2/.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package **mlr3learners** for a solid collection of essential learners.
- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Package **mlr3viz** for some generic visualizations.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: LearnerFcst, mlr_learners_fcst.adam, mlr_learners_fcst.ar, mlr_learners_fcst.arfima, mlr_learners_fcst.arima, mlr_learners_fcst.auto_adam, mlr_learners_fcst.auto_arima, mlr_learners_fcst.auto_ces, mlr_learners_fcst.auto_gum, mlr_learners_fcst.auto_msarima, mlr_learners_fcst.auto_ssarima, mlr_learners_fcst.bagged, mlr_learners_fcst.bats, mlr_learners_fcst.ces, mlr_learners_fcst.croston, mlr_learners_fcst.elm, mlr_learners_fcst.es, mlr_learners_fcst.ets, mlr_learners_fcst.gum, mlr_learners_fcst.holt_winters, mlr_learners_fcst.mean, mlr_learners_fcst.mlp, mlr_learners_fcst.msarima, mlr_learners_fcst.nnetar, mlr_learners_fcst.prophet, mlr_learners_fcst.random_walk, mlr_learners_fcst.rlgt, mlr_learners_fcst.sma, mlr_learners_fcst.sparma, mlr_learners_fcst.spline, mlr_learners_fcst.ssarima, mlr_learners_fcst.stlm, mlr_learners_fcst.struct_t, mlr_learners_fcst.tbats, mlr_learners_fcst.theta, mlr_learners_fcst.tscount

Examples

```
# Define the Learner and set parameter values
learner = lrn("fcst.tslm")
print(learner)

# Define a Task
task = tsk("airpassengers")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Print the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_measures_fcst.acf1`

Autocorrelation at Lag 1

Description

Measures the autocorrelation of the forecast residuals at lag 1. Values close to zero indicate that residuals are uncorrelated, while values far from zero suggest the model is not capturing all available information.

Details

Computed as the sample autocorrelation of the residuals at lag 1 using `stats::acf()`.

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.acf1")
msr("fcst.acf1")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: "regr"
- Range: $[-1, 1]$
- Minimize: NA
- Average: macro
- Required Prediction: "response"
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Empty ParamSet

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasureACF1`

Methods

Public methods:

- `MeasureACF1$new()`
- `MeasureACF1$clone()`

`MeasureACF1$new()`: Creates a new instance of this R6 class.

Usage:

```
MeasureACF1$new()
```

`MeasureACF1$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MeasureACF1$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- `as.data.table(mlr_measures)` for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: `mlr_measures_fcst.coverage`, `mlr_measures_fcst.mase`, `mlr_measures_fcst.mda`, `mlr_measures_fcst.mdpv`, `mlr_measures_fcst.mdv`, `mlr_measures_fcst.mpe`, `mlr_measures_fcst.msis`, `mlr_measures_fcst.pinball`, `mlr_measures_fcst.rmsse`, `mlr_measures_fcst.wape`, `mlr_measures_fcst.winkler`

`mlr_measures_fcst.coverage` *Empirical Coverage*

Description

Measures the proportion of true values that fall within the prediction interval. A well-calibrated prediction interval at level $1 - \alpha$ should have coverage close to $1 - \alpha$.

Details

$$\text{Coverage} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{l_i \leq y_i \leq u_i\}$$

where l_i and u_i are the lower and upper bounds of the prediction interval and y_i is the observed value.

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.coverage")
msr("fcst.coverage")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: "regr"
- Range: $[0, 1]$
- Minimize: NA
- Average: macro
- Required Prediction: "quantiles"
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Id	Type	Default	Range
alpha	numeric	-	$[0, 1]$

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasureCoverage`

Methods

Public methods:

- `MeasureCoverage$new()`
- `MeasureCoverage$clone()`

`MeasureCoverage$new()`: Creates a new instance of this R6 class.

Usage:

```
MeasureCoverage$new()
```

`MeasureCoverage$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MeasureCoverage$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- `as.data.table(mlr_measures)` for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: `mlr_measures_fcst.acf1`, `mlr_measures_fcst.mase`, `mlr_measures_fcst.mda`, `mlr_measures_fcst.mdpv`, `mlr_measures_fcst.mdv`, `mlr_measures_fcst.mpe`, `mlr_measures_fcst.msis`, `mlr_measures_fcst.pinball`, `mlr_measures_fcst.rmsse`, `mlr_measures_fcst.wape`, `mlr_measures_fcst.winkler`

`mlr_measures_fcst.mase`

Mean Absolute Scaled Error

Description

Measures the mean absolute error of the forecast scaled by the in-sample mean absolute error of the naive (or seasonal naive) forecast. Values less than one indicate the forecast is better than the naive baseline.

Details

$$\text{MASE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{\frac{1}{T-m}\sum_{t=m+1}^{T}|z_t - z_{t-m}|}$$

where z is the training series, m is the seasonal period, and T is the length of the training series. If `period` is `NULL` (default), the seasonal period is derived from `task$freq` and rounded to the nearest positive integer, falling back to one when the task frequency is unavailable.

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.mase")
msr("fcst.mase")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: “regr”
- Range: $[0, \infty)$
- Minimize: TRUE
- Average: macro
- Required Prediction: “response”
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Id	Type	Default	Range
period	integer	NULL	$[1, \infty)$

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasureMASE`

Methods

Public methods:

- `MeasureMASE$new()`
- `MeasureMASE$clone()`

`MeasureMASE$new()`: Creates a new instance of this R6 class.

Usage:

```
MeasureMASE$new()
```

`MeasureMASE$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MeasureMASE$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Hyndman RJ, Koehler AB (2006). "Another look at measures of forecast accuracy." *International Journal of Forecasting*, **22**(4), 679–688.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval`
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- `as.data.table(mlr_measures)` for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: `mlr_measures_fcst.acf1`, `mlr_measures_fcst.coverage`, `mlr_measures_fcst.mda`, `mlr_measures_fcst.mdpv`, `mlr_measures_fcst.mdv`, `mlr_measures_fcst.mpe`, `mlr_measures_fcst.msis`, `mlr_measures_fcst.pinball`, `mlr_measures_fcst.rmsse`, `mlr_measures_fcst.wape`, `mlr_measures_fcst.winkler`

`mlr_measures_fcst.mda` *Mean Directional Accuracy*

Description

Measure of the proportion of correctly predicted directions between successive observations in forecast tasks.

Details

$$\text{MDA} = (a - b)\,\frac{1}{n-1}\sum_{i=2}^{n}\mathbf{1}\{\text{sign}(y_i - y_{i-1}) = \text{sign}(\hat{y}_i - y_{i-1})\} \;+\; b$$

where `a` is the reward for a correct direction (default `1`), `b` is the penalty for an incorrect direction (default `0`), and `n` is the number of observations.

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.mda")
msr("fcst.mda")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: "regr"
- Range: $(-\infty, \infty)$
- Minimize: FALSE
- Average: macro
- Required Prediction: "response"
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Id	Type	Default	Range
reward	numeric	-	$(-\infty, \infty)$
penalty	numeric	-	$(-\infty, \infty)$

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasureMDA`

Methods

Public methods:

- `MeasureMDA$new()`
- `MeasureMDA$clone()`

`MeasureMDA$new()`: Creates a new instance of this R6 class.

Usage:

```
MeasureMDA$new()
```

`MeasureMDA$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MeasureMDA$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Blaskowitz O, Herwartz H (2011). "On economic evaluation of directional forecasts." *International Journal of Forecasting*, **27**(4), 1058–1065.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- `as.data.table(mlr_measures)` for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: `mlr_measures_fcst.acf1`, `mlr_measures_fcst.coverage`, `mlr_measures_fcst.mase`, `mlr_measures_fcst.mdpv`, `mlr_measures_fcst.mdv`, `mlr_measures_fcst.mpe`, `mlr_measures_fcst.msis`, `mlr_measures_fcst.pinball`, `mlr_measures_fcst.rmsse`, `mlr_measures_fcst.wape`, `mlr_measures_fcst.winkler`

`mlr_measures_fcst.mdpv`

Mean Directional Percentage Value

Description

Measure of average percentage-weighted directional accuracy in forecast tasks.

Details

$$\text{MDPV} = \frac{1}{n-1}\sum_{i=2}^{n}\left|\frac{y_i - y_{i-1}}{y_{i-1}}\right| \times \begin{cases} +1, & \text{if } \operatorname{sign}(y_i - y_{i-1}) = \operatorname{sign}(\hat{y}_i - y_{i-1}), \\ -1, & \text{otherwise.} \end{cases}$$

where `n` is the number of observations.

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.mdpv")
msr("fcst.mdpv")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: "regr"
- Range: $(-\infty, \infty)$
- Minimize: FALSE
- Average: macro
- Required Prediction: "response"
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Empty ParamSet

Super classes

mlr3::Measure -> mlr3::MeasureRegr -> MeasureMDPV

Methods

Public methods:

- MeasureMDPV$new()
- MeasureMDPV$clone()

MeasureMDPV$new(): Creates a new instance of this R6 class.

Usage:

```
MeasureMDPV$new()
```

MeasureMDPV$clone(): The objects of this class are cloneable with this method.

Usage:

```
MeasureMDPV$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Blaskowitz O, Herwartz H (2011). "On economic evaluation of directional forecasts." *International Journal of Forecasting*, **27**(4), 1058–1065.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- as.data.table(mlr_measures) for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: mlr_measures_fcst.acf1, mlr_measures_fcst.coverage, mlr_measures_fcst.mase, mlr_measures_fcst.mda, mlr_measures_fcst.mdv, mlr_measures_fcst.mpe, mlr_measures_fcst.msis, mlr_measures_fcst.pinball, mlr_measures_fcst.rmsse, mlr_measures_fcst.wape, mlr_measures_fcst.winkler

`mlr_measures_fcst.mdv` *Mean Directional Value*

Description

Measure of average magnitude-weighted directional accuracy in forecast tasks.

Details

$$\text{MDV} = \frac{1}{n-1}\sum_{i=2}^{n}|y_i - y_{i-1}| \times \begin{cases} +1, & \text{if } \text{sign}(y_i - y_{i-1}) = \text{sign}(\hat{y}_i - y_{i-1}), \\ -1, & \text{otherwise.} \end{cases}$$

where `n` is the number of observations.

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.mdv")
msr("fcst.mdv")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: "regr"
- Range: $(-\infty, \infty)$
- Minimize: FALSE
- Average: macro
- Required Prediction: "response"
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Empty ParamSet

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasureMDV`

Methods

Public methods:

- `MeasureMDV$new()`
- `MeasureMDV$clone()`

`MeasureMDV$new()`: Creates a new instance of this R6 class.

Usage:

```
MeasureMDV$new()
```

`MeasureMDV$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MeasureMDV$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Blaskowitz O, Herwartz H (2011). "On economic evaluation of directional forecasts." *International Journal of Forecasting*, **27**(4), 1058–1065.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- `as.data.table(mlr_measures)` for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: `mlr_measures_fcst.acf1`, `mlr_measures_fcst.coverage`, `mlr_measures_fcst.mase`, `mlr_measures_fcst.mda`, `mlr_measures_fcst.mdpv`, `mlr_measures_fcst.mpe`, `mlr_measures_fcst.msis`, `mlr_measures_fcst.pinball`, `mlr_measures_fcst.rmsse`, `mlr_measures_fcst.wape`, `mlr_measures_fcst.winkler`

`mlr_measures_fcst.mpe` *Mean Percentage Error*

Description

Measure of the average signed percentage error of forecasts. Positive values indicate systematic under-forecasting, negative values indicate over-forecasting.

Details

$$\text{MPE} = \frac{100}{n} \sum_{i=1}^{n} \frac{y_i - \hat{y}_i}{y_i}$$

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.mpe")
msr("fcst.mpe")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: "regr"
- Range: $(-\infty, \infty)$
- Minimize: NA
- Average: macro
- Required Prediction: "response"
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Empty ParamSet

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasureMPE`

Methods

Public methods:

- `MeasureMPE$new()`
- `MeasureMPE$clone()`

`MeasureMPE$new()`: Creates a new instance of this R6 class.

Usage:

```
MeasureMPE$new()
```

`MeasureMPE$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MeasureMPE$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- `as.data.table(mlr_measures)` for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: `mlr_measures_fcst.acf1`, `mlr_measures_fcst.coverage`, `mlr_measures_fcst.mase`, `mlr_measures_fcst.mda`, `mlr_measures_fcst.mdpv`, `mlr_measures_fcst.mdv`, `mlr_measures_fcst.msis`, `mlr_measures_fcst.pinball`, `mlr_measures_fcst.rmsse`, `mlr_measures_fcst.wape`, `mlr_measures_fcst.winkler`

---

mlr_measures_fcst.msis

Mean Scaled Interval Score

Description

Measures the quality of central prediction intervals, scaling the interval (Winkler) score by the in-sample mean absolute error of the naive (or seasonal naive) forecast. The interval score rewards narrow intervals and penalizes observations falling outside them, and the scaling makes the measure comparable across series of different magnitudes. This is the prediction-interval metric used in the M4 competition. Smaller scores indicate better calibrated and narrower intervals.

Details

For a central interval at level `1 - alpha` with lower and upper bounds l_i and u_i (the `alpha/2` and `1 - alpha/2` quantiles):

$$\text{MSIS} = \frac{\frac{1}{n}\sum_{i=1}^{n}(u_i - l_i) + \frac{2}{\alpha}(l_i - y_i)\mathbf{1}\{y_i < l_i\} + \frac{2}{\alpha}(y_i - u_i)\mathbf{1}\{y_i > u_i\}}{\frac{1}{T-m}\sum_{t=m+1}^{T}|z_t - z_{t-m}|}$$

where z is the training series, m is the seasonal period, and T is the length of the training series. For keyed tasks the score is computed per series and averaged. If `period` is `NULL` (default), the seasonal period is derived from `task$freq` and rounded to the nearest positive integer, falling back to one when the task frequency is unavailable.

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.msis")
msr("fcst.msis")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: "regr"
- Range: $[0, \infty)$
- Minimize: TRUE
- Average: macro
- Required Prediction: "quantiles"
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Id	Type	Default	Range
alpha	numeric	-	$[0, 1]$
period	integer	NULL	$[1, \infty)$

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasureMSIS`

Methods

Public methods:

- `MeasureMSIS$new()`
- `MeasureMSIS$clone()`

`MeasureMSIS$new()`: Creates a new instance of this R6 class.

Usage:

```
MeasureMSIS$new()
```

`MeasureMSIS$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MeasureMSIS$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Gneiting T, Raftery AE (2007). "Strictly Proper Scoring Rules, Prediction, and Estimation." *Journal of the American Statistical Association*, **102**(477), 359–378.

Makridakis S, Spiliotis E, Assimakopoulos V (2020). "The M4 Competition: 100,000 time series and 61 forecasting methods." *International Journal of Forecasting*, **36**(1), 54–74.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval`
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- `as.data.table(mlr_measures)` for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: `mlr_measures_fcst.acf1`, `mlr_measures_fcst.coverage`, `mlr_measures_fcst.mase`, `mlr_measures_fcst.mda`, `mlr_measures_fcst.mdpv`, `mlr_measures_fcst.mdv`, `mlr_measures_fcst.mpe`, `mlr_measures_fcst.pinball`, `mlr_measures_fcst.rmsse`, `mlr_measures_fcst.wape`, `mlr_measures_fcst.winkler`

`mlr_measures_fcst.pinball` *Pinball Loss*

Description

Measures the quality of quantile (probabilistic) forecasts using the pinball loss, also known as the quantile loss. The loss is averaged over all observations and all predicted quantile levels. Smaller scores indicate better calibrated quantile forecasts.

Details

For a single quantile level τ with forecast q_i and observation y_i the pinball loss is

$$L_\tau(y_i, q_i) = \begin{cases} \tau\,(y_i - q_i), & \text{if } y_i \geq q_i \\ (1-\tau)\,(q_i - y_i), & \text{if } y_i < q_i \end{cases}$$

The reported score is twice the mean of L_τ over all observations and all quantile levels τ, matching the convention used by **fabletools** so that the median ($\tau = 0.5$) pinball loss equals the mean absolute error.

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.pinball")
msr("fcst.pinball")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: “regr”
- Range: $[0, \infty)$
- Minimize: TRUE
- Average: macro
- Required Prediction: “quantiles”
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Empty ParamSet

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasurePinball`

Methods

Public methods:

- `MeasurePinball$new()`
- `MeasurePinball$clone()`

`MeasurePinball$new()`: Creates a new instance of this R6 class.

Usage:

```
MeasurePinball$new()
```

`MeasurePinball$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MeasurePinball$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Koenker R, Bassett G (1978). "Regression Quantiles." *Econometrica*, **46**(1), 33–50.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- `as.data.table(mlr_measures)` for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: `mlr_measures_fcst.acf1`, `mlr_measures_fcst.coverage`, `mlr_measures_fcst.mase`, `mlr_measures_fcst.mda`, `mlr_measures_fcst.mdpv`, `mlr_measures_fcst.mdv`, `mlr_measures_fcst.mpe`, `mlr_measures_fcst.msis`, `mlr_measures_fcst.rmsse`, `mlr_measures_fcst.wape`, `mlr_measures_fcst.winkler`

`mlr_measures_fcst.rmsse`

Root Mean Squared Scaled Error

Description

Measures the root mean squared error of the forecast scaled by the in-sample mean squared error of the naive (or seasonal naive) forecast. Values less than one indicate the forecast is better than the naive baseline.

Details

$$\text{RMSSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\frac{(y_i - \hat{y}_i)^2}{\frac{1}{T-m}\sum_{t=m+1}^{T}(z_t - z_{t-m})^2}}$$

where z is the training series, m is the seasonal period, and T is the length of the training series. If `period` is `NULL` (default), the seasonal period is derived from `task$freq` and rounded to the nearest positive integer, falling back to one when the task frequency is unavailable.

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.rmsse")
msr("fcst.rmsse")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: "regr"
- Range: $[0, \infty)$
- Minimize: TRUE
- Average: macro
- Required Prediction: "response"
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Id	Type	Default	Range
period	integer	NULL	$[1, \infty)$

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasureRMSSE`

Methods

Public methods:

- `MeasureRMSSE$new()`
- `MeasureRMSSE$clone()`

`MeasureRMSSE$new()`: Creates a new instance of this R6 class.

Usage:

```
MeasureRMSSE$new()
```

`MeasureRMSSE$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MeasureRMSSE$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Hyndman RJ, Koehler AB (2006). "Another look at measures of forecast accuracy." *International Journal of Forecasting*, **22**(4), 679–688.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval`
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- `as.data.table(mlr_measures)` for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: `mlr_measures_fcst.acf1`, `mlr_measures_fcst.coverage`, `mlr_measures_fcst.mase`, `mlr_measures_fcst.mda`, `mlr_measures_fcst.mdpv`, `mlr_measures_fcst.mdv`, `mlr_measures_fcst.mpe`, `mlr_measures_fcst.msis`, `mlr_measures_fcst.pinball`, `mlr_measures_fcst.wape`, `mlr_measures_fcst.winkler`

`mlr_measures_fcst.wape`

Weighted Absolute Percentage Error

Description

Measure of the total absolute error of forecasts as a percentage of the total absolute truth. It weights each error by the magnitude of the series, making it robust to individual observations close to zero where the ordinary percentage error is undefined.

Details

$$\text{WAPE} = 100 \cdot \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{\sum_{i=1}^{n} |y_i|}$$

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.wape")
msr("fcst.wape")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: "regr"
- Range: $[0, \infty)$
- Minimize: TRUE
- Average: macro
- Required Prediction: "response"
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Empty ParamSet

Super classes

mlr3::Measure -> mlr3::MeasureRegr -> MeasureWAPE

Methods

Public methods:

- MeasureWAPE$new()
- MeasureWAPE$clone()

MeasureWAPE$new(): Creates a new instance of this R6 class.

Usage:

```
MeasureWAPE$new()
```

MeasureWAPE$clone(): The objects of this class are cloneable with this method.

Usage:

```
MeasureWAPE$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- as.data.table(mlr_measures) for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: mlr_measures_fcst.acf1, mlr_measures_fcst.coverage, mlr_measures_fcst.mase, mlr_measures_fcst.mda, mlr_measures_fcst.mdpv, mlr_measures_fcst.mdv, mlr_measures_fcst.mpe, mlr_measures_fcst.msis, mlr_measures_fcst.pinball, mlr_measures_fcst.rmsse, mlr_measures_fcst.winkler

`mlr_measures_fcst.winkler` *Winkler Score*

Description

Measures the quality of prediction intervals by combining their width with a penalty for observations falling outside the interval. Smaller scores indicate better calibrated and narrower intervals.

Details

$$W_i = \begin{cases} (u_i - l_i) + \frac{2}{\alpha}(l_i - y_i), & \text{if } y_i < l_i \\ (u_i - l_i), & \text{if } l_i \leq y_i \leq u_i \\ (u_i - l_i) + \frac{2}{\alpha}(y_i - u_i), & \text{if } y_i > u_i \end{cases}$$

where l_i and u_i are the lower and upper bounds of the prediction interval, y_i is the observed value, and $\alpha = 1 - \text{level}/100$ is the significance level. The Winkler score is then the mean of W_i over all observations.

Dictionary

This mlr3::Measure can be instantiated via the dictionary mlr3::mlr_measures or with the associated sugar function `mlr3::msr()`:

```
mlr_measures$get("fcst.winkler")
msr("fcst.winkler")
```

Task type

Forecast measures are registered with `task_type = "regr"` so they compose with the standard regression measures (e.g. mlr3::mlr_measures_regr.rmse) on the PredictionFcst that forecast learners produce. List them via the key prefix, not the task type, as the latter returns nothing:

```
as.data.table(mlr_measures)[grepl("^fcst", key)]
```

Meta Information

- Task type: “regr”
- Range: $[0, \infty)$
- Minimize: TRUE
- Average: macro
- Required Prediction: “quantiles”
- Required Packages: **mlr3**, **mlr3forecast**

Parameters

Id	Type	Default	Range
alpha	numeric	-	$[0, 1]$

Super classes

`mlr3::Measure` -> `mlr3::MeasureRegr` -> `MeasureWinkler`

Methods

Public methods:

- `MeasureWinkler$new()`
- `MeasureWinkler$clone()`

`MeasureWinkler$new()`: Creates a new instance of this R6 class.

Usage:

```
MeasureWinkler$new()
```

`MeasureWinkler$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MeasureWinkler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Winkler RL (1972). "A Decision-Theoretic Approach to Interval Estimation." *Journal of the American Statistical Association*, **67**(337), 187–191.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-eval
- Package **mlr3measures** for the scoring functions.
- Dictionary of Measures: mlr3::mlr_measures
- `as.data.table(mlr_measures)` for a table of available Measures in the running session (depending on the loaded packages).
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

Other Measure: `mlr_measures_fcst.acf1`, `mlr_measures_fcst.coverage`, `mlr_measures_fcst.mase`, `mlr_measures_fcst.mda`, `mlr_measures_fcst.mdpv`, `mlr_measures_fcst.mdv`, `mlr_measures_fcst.mpe`, `mlr_measures_fcst.msis`, `mlr_measures_fcst.pinball`, `mlr_measures_fcst.rmsse`, `mlr_measures_fcst.wape`

`mlr_pipeops_fcst.avg` *Weighted Prediction Averaging for Forecasts*

Description

Performs (weighted) averaging of forecast PredictionFcsts, mirroring mlr3pipelines::PipeOpRegrAvg but preserving the forecast prediction type, which plain `regravg` would drop. The output is a PredictionFcst that keeps the time index and key columns (carried in the `extra` slot with explicit column roles), so `$order`, `$key`, `autoplot.PredictionFcst()`, and forecast `task_type` inference keep working through the ensemble.

Connect it to several PipeOpLearner outputs (classical forecast learners or RecursiveForecaster / DirectForecaster) to average their forecasts.

Parameters

The parameters are the parameters inherited from mlr3pipelines::PipeOpRegrAvg.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpEnsemble` -> `mlr3pipelines::PipeOpRegrAvg` -> `PipeOpFcstAvg`

Methods

Public methods:

- `PipeOpFcstAvg$new()`
- `PipeOpFcstAvg$clone()`

`PipeOpFcstAvg$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFcstAvg$new(
  innum = 0L,
  collect_multiplicity = FALSE,
  id = "fcst.avg",
  param_vals = list()
)
```

Arguments:

`innum (numeric(1))`
: Number of input channels. Default `0` creates a vararg channel taking an arbitrary number of inputs.

`collect_multiplicity (logical(1))`
: If `TRUE`, the single input is a Multiplicity collecting channel. Requires `innum = 0`. Default `FALSE`.

`id (character(1))`
: Identifier of resulting object, default `"fcst.avg"`.

`param_vals` (named `list()`)
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFcstAvg$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFcstAvg$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
graph = gunion(list(
  po("learner", lrn("fcst.auto_arima"), id = "arima"),
  po("learner", lrn("fcst.ets"), id = "ets")
)) %>>%
  po("fcst.avg")
flrn = as_learner(graph)$train(task)
forecast(flrn, task, 12L)
```

---

mlr_pipeops_fcst.catch22

Time Series Feature Extraction (catch22)

Description

This `PipeOp` extracts the 22 (or 24) canonical time series characteristics (catch22) from the target variable. For more details, see `Rcatch22::catch22_all()`, which is called internally on the ordered target vector.

For other time series feature extractors, see PipeOpFcstTsfeats and PipeOpFcstFeasts.

Parameters

The parameters are the parameters inherited from mlr3pipelines::PipeOpTaskPreproc, as well as:

- `catch24` :: `logical(1)`
  If `TRUE`, additionally compute the mean and standard deviation (the catch24 set). Default `FALSE`.

Naming

The new columns are named `{target}_catch22_{feature}`. If the target was called `"y"` and the feature is `"DN_HistogramMode_5"`, the corresponding new column will be called `"y_catch22_DN_HistogramMode_5"`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `PipeOpFcstCatch22`

Methods

Public methods:

- `PipeOpFcstCatch22$new()`
- `PipeOpFcstCatch22$clone()`

`PipeOpFcstCatch22$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFcstCatch22$new(id = "fcst.catch22", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fcst.catch22"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFcstCatch22$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFcstCatch22$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
po = po("fcst.catch22")
out = po$train(list(task))[[1L]]
out$head()
```

---

mlr_pipeops_fcst.feasts

Time Series Feature Extraction (feasts)

Description

Computes per-series summary features from the target variable via `fabletools::features()` with feature functions from the **feasts** package, and broadcasts them as constant columns to every row of the corresponding series. For an unkeyed task the features are broadcast to every row. For a keyed task each key contributes one feature vector.

This is the **feasts** (tidyverts) counterpart of PipeOpFcstTsfeats. Predicting on a key that was not seen during training is an error.

Parameters

The parameters are the parameters inherited from mlr3pipelines::PipeOpTaskPreproc, as well as:

- `features` :: `list()`
  A list of **feasts** feature functions (e.g. feasts::feat_acf, feasts::feat_stl) or a `fabletools::feature_set()`.
  Default `list(feasts::feat_acf, feasts::feat_stl)`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `PipeOpFcstFeasts`

Methods

Public methods:

- `PipeOpFcstFeasts$new()`
- `PipeOpFcstFeasts$clone()`

`PipeOpFcstFeasts$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFcstFeasts$new(id = "fcst.feasts", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fcst.feasts"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFcstFeasts$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFcstFeasts$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
po = po("fcst.feasts", features = list(feasts::feat_acf))
out = po$train(list(task))[[1L]]
out$head()

# select features by tag via fabletools::feature_set() (requires feasts to be attached so its
# feature registry is populated)
library(feasts)
features = fabletools::feature_set(pkgs = "feasts", tags = "autocorrelation")
po = po("fcst.feasts", features = features)
po$train(list(task))[[1L]]$head()
```

`mlr_pipeops_fcst.fourier`

Create Fourier Features for Seasonality

Description

Creates pairs of Fourier (harmonic) terms `sin(2 * pi * k * t / period)` and `cos(2 * pi * k * t / period)` as new feature columns, for `k = 1, ..., K` harmonics per seasonal `period`, where `t` is the per-series time position. They encode seasonality as a flexible alternative to seasonal lags, in particular for long or non-integer periods and multiple seasonalities at once.

Parameters

The parameters are the parameters inherited from mlr3pipelines::PipeOpTaskPreprocSimple, as well as the following parameters:

- `period` :: `numeric()` | `NULL`
  Seasonal period(s), in number of observations per cycle. May be non-integer and may contain multiple periods for multiple seasonalities. If `NULL` (default), the period is derived from the task's frequency (`task$freq`).
- `K` :: `integer()`
  Number of Fourier harmonics per `period`. Either a single value recycled to all periods, or one value per period. Each `K` must satisfy `2 * K <= period`. Default `1L`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `mlr3pipelines::PipeOpTaskPreprocSimple` -> `PipeOpFcstFourier`

Methods

Public methods:

- `PipeOpFcstFourier$new()`
- `PipeOpFcstFourier$clone()`

`PipeOpFcstFourier$new()`**:** Initializes a new instance of this Class.

Usage:

```
PipeOpFcstFourier$new(id = "fcst.fourier", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fcst.fourier"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFcstFourier$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PipeOpFcstFourier$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

De Livera AM, Hyndman RJ, Snyder RD (2011). "Forecasting time series with complex seasonal patterns using exponential smoothing." *Journal of the American Statistical Association*, **106**(496), 1513–1527.

Hyndman RJ, Khandakar Y (2008). "Automatic Time Series Forecasting: The forecast Package for R." *Journal of Statistical Software*, **27**(3), 1–22. doi:10.18637/jss.v027.i03.

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
po = po("fcst.fourier", period = 12, K = 3L)
new_task = po$train(list(task))[[1L]]
new_task$head()
```

`mlr_pipeops_fcst.lags` *Create Lags of Target Variable*

Description

Creates lagged versions of the target variable as new feature columns.

At train time the first rows of each series have no history for the requested lags. These incomplete rows are dropped (the autoregressive-fit convention), so the base learner never sees `NA` lags. A keyed series shorter than the largest lag is dropped entirely, with a warning.

At predict time lags are computed from the full series history.

Parameters

The parameters are the parameters inherited from mlr3pipelines::PipeOpTaskPreproc, as well as the following parameters:

- `lags` :: `integer()`
  The lags to create.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `PipeOpFcstLags`

Methods

Public methods:

- PipeOpFcstLags$new()
- PipeOpFcstLags$clone()

`PipeOpFcstLags$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFcstLags$new(id = "fcst.lags", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fcst.lags"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFcstLags$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFcstLags$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
po = po("fcst.lags", lags = 1:3)
new_task = po$train(list(task))[[1L]]
new_task$head()
```

---

mlr_pipeops_fcst.rolling

Create Rolling Window Features of Target Variable

Description

Creates rolling-window summary statistics of the target variable as new feature columns. The window ends at position `t - lag` (exclusive of the current and `lag - 1` most recent values) and has size `window_size`. Use `window_size = Inf` for an expanding window that grows to include all history up to `t - lag`.

At train time rows whose window has insufficient history are `NA` and are dropped, matching PipeOpFcstLags. Predict keeps all rows.

At predict time rolling features are computed from the full series history.

Parameters

The parameters are the parameters inherited from mlr3pipelines::PipeOpTaskPreproc, as well as the following parameters:

- `funs` :: `character()`
  Aggregation functions. Subset of `c("mean", "median", "sd", "min", "max", "sum")`. Default `"mean"`.
- `window_sizes` :: `numeric()`
  Window sizes. Every combination of `funs` and `window_sizes` produces one output column. Finite sizes must be whole numbers. `Inf` requests an expanding window (all history up to `t - lag`). Default `3L`.
- `lag` :: `integer(1)`
  Minimum lag before the window starts. Must be `>= 1` to avoid leakage. Default `1L`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `PipeOpFcstRolling`

Methods

Public methods:

- `PipeOpFcstRolling$new()`
- `PipeOpFcstRolling$clone()`

`PipeOpFcstRolling$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFcstRolling$new(id = "fcst.rolling", param_vals = list())
```

Arguments:

`id` (`character(1)`)
: Identifier of resulting object, default `"fcst.rolling"`.

`param_vals` (named `list()`)
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFcstRolling$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFcstRolling$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
po = po("fcst.rolling", funs = c("mean", "sd"), window_sizes = c(3L, 12L))
new_task = po$train(list(task))[[1L]]
new_task$head()
```

`mlr_pipeops_fcst.splitkey` *Split a Forecast Task into Per-Series Tasks*

Description

Splits a keyed (multi-series) TaskFcst into a Multiplicity of single-series tasks, one per key combination. Subsequent PipeOps are executed once per series until a `po("fcst.unitekey")` is reached, fitting one local model per series instead of one global model pooled across series.

The per-series tasks carry no key columns, so classical univariate learners (e.g. `lrn("fcst.ets")`) compose as well. The key groups observed during training are stored in the `$state` and the task must contain exactly the same key groups at predict time.

Parameters

This PipeOp has no parameters.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpFcstSplitKey`

Methods

Public methods:

- `PipeOpFcstSplitKey$new()`
- `PipeOpFcstSplitKey$clone()`

`PipeOpFcstSplitKey$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFcstSplitKey$new(id = "fcst.splitkey", param_vals = list())
```

Arguments:

`id` (`character(1)`)
: Identifier of resulting object, default `"fcst.splitkey"`.

`param_vals` (named `list()`)
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFcstSplitKey$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFcstSplitKey$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3pipelines)
library(data.table)
dt = CJ(
  month = seq(as.Date("2024-01-01"), by = "month", length.out = 36L),
  id = factor(c("a", "b"))
)
dt[, value := rnorm(.N, mean = fifelse(id == "a", 10, 20))]
task = as_task_fcst(dt, target = "value", order = "month", key = "id", freq = "month")
graph = po("fcst.splitkey") %>>% lrn("fcst.ets") %>>% po("fcst.unitekey")
flrn = as_learner(graph)$train(task)
forecast(flrn, task, 12L)
```

`mlr_pipeops_fcst.targetboxcox`

Box-Cox Transform the Target Variable

Description

Applies a Box-Cox transformation to the target variable to stabilize the variance, producing the new target `BoxCox(y, lambda)`. The transformation is pointwise and monotonic, so no rows are dropped and predictions are inverted via `forecast::InvBoxCox()`. `lambda = 0` is the log transformation. When `lambda` is `NULL` (default) it is estimated from the training data, per series on keyed tasks. Predicting a series not seen during training is an error.

Box-Cox and log transformations require strictly positive target values. Non-positive values produce `NaN` or an error. A negative estimated `lambda` can make `forecast::InvBoxCox()` return `NA` for upper quantiles. Set `lower = 0` to avoid this.

Parameters

The parameters are the parameters inherited from mlr3pipelines::PipeOpTargetTrafo, as well as the following:

- `lambda` :: `numeric(1)` | `NULL`
  Box-Cox transformation parameter. `NULL` (default) estimates it from the training data, `0` is the log transformation, any other numeric is used as a fixed value.
- `method` :: `character(1)`
  Method used to estimate `lambda` when `lambda = NULL`, one of `"guerrero"` (default) or `"loglik"`. See `forecast::BoxCox.lambda()`.
- `lower` :: `numeric(1)`
  Lower bound for the estimated `lambda`. Default `-1`.
- `upper` :: `numeric(1)`
  Upper bound for the estimated `lambda`. Default `2`.

Limitations

This PipeOp must not be placed *inside* a RecursiveForecaster or DirectForecaster graph and is rejected at construction. Use it inside a plain mlr3pipelines::GraphLearner via `ppl("targettrafo", ...)`, or wrap the forecaster itself with `ppl("targettrafo", ...)` so all horizons are inverted together.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTargetTrafo` -> `PipeOpTargetTrafoBoxCox`

Methods

Public methods:

- `PipeOpTargetTrafoBoxCox$new()`
- `PipeOpTargetTrafoBoxCox$clone()`

`PipeOpTargetTrafoBoxCox$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpTargetTrafoBoxCox$new(id = "fcst.targetboxcox", param_vals = list())
```

Arguments:

`id` (`character(1)`)
: Identifier of resulting object, default `"fcst.targetboxcox"`.

`param_vals` (named `list()`)
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpTargetTrafoBoxCox$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpTargetTrafoBoxCox$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
split = partition(task, ratio = 0.8)
flrn = as_learner(ppl("targettrafo",
 graph = DirectForecaster$new(lrn("regr.rpart"), lags = 1:3, horizons = length(split$test)),
  trafo_pipeop = po("fcst.targetboxcox")
))
flrn$train(task, split$train)
flrn$predict(task, split$test)
```

`mlr_pipeops_fcst.targetdiff` *Difference the Target Variable*

Description

Differences the target variable with lag `lag`, producing the new target `y'_t = y_t - y_{t - lag}`. The first `lag` rows are dropped during training and predictions are inverted back to the original scale. On keyed (multi-series) tasks this happens within each series. Series too short for the requested lag are dropped with a warning, and predicting a series not seen during training is an error.

Use `lag = 1` to remove a trend and `lag = 12` (or the seasonal period) to remove seasonality.

Parameters

The parameters are the parameters inherited from mlr3pipelines::PipeOpTargetTrafo, as well as the following:

- `lag` :: `integer(1)`
  Lag to difference at. Default `1L`.

Limitations

This PipeOp must not be placed *inside* a RecursiveForecaster or DirectForecaster graph and is rejected at construction. Use it inside a plain mlr3pipelines::GraphLearner via `ppl("targettrafo", ...)`, or wrap the forecaster itself with `ppl("targettrafo", ...)` so all horizons are inverted together.

Quantile predictions cannot be inverted and are rejected.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTargetTrafo` -> `PipeOpTargetTrafoDifference`

Methods

Public methods:

- `PipeOpTargetTrafoDifference$new()`
- `PipeOpTargetTrafoDifference$clone()`

`PipeOpTargetTrafoDifference$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpTargetTrafoDifference$new(id = "fcst.targetdiff", param_vals = list())
```

Arguments:

`id (character(1))`
Identifier of resulting object, default `"fcst.targetdiff"`.

param_vals (named list())
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default list().

PipeOpTargetTrafoDifference$clone(): The objects of this class are cloneable with this method.

Usage:

```
PipeOpTargetTrafoDifference$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
split = partition(task, ratio = 0.8)
flrn = as_learner(ppl("targettrafo",
 graph = DirectForecaster$new(lrn("regr.rpart"), lags = 1:3, horizons = length(split$test)),
  trafo_pipeop = po("fcst.targetdiff", lag = 1L)
))
flrn$train(task, split$train)
flrn$predict(task, split$test)
```

---

mlr_pipeops_fcst.tsfeats

Time Series Feature Extraction

Description

Computes per-series summary features from the target variable via tsfeatures::tsfeatures() and broadcasts them as constant columns to every row of the corresponding series. For an unkeyed task the features are broadcast to every row. For a keyed task each key contributes one feature vector.

Predicting on a key that was not seen during training is an error.

Parameters

The parameters are the parameters inherited from mlr3pipelines::PipeOpTaskPreproc, as well as:

- features :: character()
  Function names from the tsfeatures namespace that return numeric feature vectors. Default c("frequency", "stl_features", "entropy", "acf_features").
- scale :: logical(1)
  If TRUE, scale each series to mean 0 and sd 1 before feature extraction. Default TRUE.
- trim :: logical(1)
  If TRUE, trim values outside ±trim_amount before feature extraction. Default FALSE.

- `trim_amount` :: `numeric(1)`
  Trimming threshold. Default `0.1`.
- `parallel` :: `logical(1)`
  If `TRUE`, compute features in parallel via a `future::plan()`. Default `FALSE`.
- `multiprocess` :: `function`
  Function from the `future` package used when `parallel = TRUE`. Default `future::multisession()`.
- `na.action` :: `function`
  Missing-value handler. Default `stats::na.pass()`.

Super classes

`mlr3pipelines::PipeOp` -> `mlr3pipelines::PipeOpTaskPreproc` -> `PipeOpFcstTsfeats`

Methods

Public methods:

- `PipeOpFcstTsfeats$new()`
- `PipeOpFcstTsfeats$clone()`

`PipeOpFcstTsfeats$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFcstTsfeats$new(id = "fcst.tsfeats", param_vals = list())
```

Arguments:

`id` (`character(1)`)
: Identifier of resulting object, default `"fcst.tsfeats"`.

`param_vals` (named `list()`)
: List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFcstTsfeats$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFcstTsfeats$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
po = po("fcst.tsfeats", features = c("entropy", "acf_features"))
out = po$train(list(task))[[1L]]
out$head()
```

`mlr_pipeops_fcst.unitekey` *Unite Per-Series Forecasts into One Prediction*

Description

Row-binds a Multiplicity of per-series PredictionFcsts, as created downstream of `po("fcst.splitkey")`, into a single PredictionFcst.

The series identity is rebuilt from the multiplicity names in the prediction's `extra` slot, and its role is stored in `col_roles`. This keeps `$key`, `as.data.table()`, and `autoplot.PredictionFcst()` working. Set `key` to the task's key column name to get predictions column-compatible with global forecasters such as RecursiveForecaster, which attach the original key column.

Parameters

- `key` :: `character(1)`
  Name of the rebuilt series-identity column in the prediction's `extra` slot. Default `"key"`.

Super class

`mlr3pipelines::PipeOp` -> `PipeOpFcstUniteKey`

Methods

Public methods:

- `PipeOpFcstUniteKey$new()`
- `PipeOpFcstUniteKey$clone()`

`PipeOpFcstUniteKey$new()`: Initializes a new instance of this Class.

Usage:

```
PipeOpFcstUniteKey$new(id = "fcst.unitekey", param_vals = list())
```

Arguments:

`id` (`character(1)`)
Identifier of resulting object, default `"fcst.unitekey"`.

`param_vals` (named `list()`)
List of hyperparameter settings, overwriting the hyperparameter settings that would otherwise be set during construction. Default `list()`.

`PipeOpFcstUniteKey$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PipeOpFcstUniteKey$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(mlr3pipelines)
library(data.table)
dt = CJ(
  month = seq(as.Date("2024-01-01"), by = "month", length.out = 36L),
  id = factor(c("a", "b"))
)
dt[, value := rnorm(.N, mean = fifelse(id == "a", 10, 20))]
task = as_task_fcst(dt, target = "value", order = "month", key = "id", freq = "month")
graph = po("fcst.splitkey") %>>% lrn("fcst.ets") %>>% po("fcst.unitekey")
flrn = as_learner(graph)$train(task)
forecast(flrn, task, 12L)
```

---

mlr_resamplings_fcst.cv

Forecast Cross-Validation Resampling

Description

Splits data using a `folds`-folds (default: 5 folds) rolling window cross-validation.

Dictionary

This Resampling can be instantiated via the dictionary mlr_resamplings or with the associated sugar function rsmp():

```
mlr_resamplings$get("fcst.cv")
rsmp("fcst.cv")
```

Parameters

- `horizon` (`integer(1)`)
  Forecasting horizon in the test sets, i.e. number of test samples for each fold.
- `folds` (`integer(1)`)
  Number of folds.
- `step_size` (`integer(1)`)
  Step size between windows.
- `window_size` (`integer(1)`)
  Size of the rolling window. For `fixed_window` = `TRUE`, this is the exact training window size. For `fixed_window` = `FALSE` (expanding window), this is the minimum number of training observations in the first fold.
- `fixed_window` (`logical(1)`)
  Should a fixed sized window be used? If `FALSE` an expanding window is used.

Super class

`mlr3::Resampling` -> `ResamplingFcstCV`

Active bindings

`iters (integer(1))`
Returns the number of resampling iterations, depending on the values stored in the `param_set`.

Methods

Public methods:

- `ResamplingFcstCV$new()`
- `ResamplingFcstCV$clone()`

`ResamplingFcstCV$new()`: Creates a new instance of this R6 class.

Usage:

```
ResamplingFcstCV$new()
```

`ResamplingFcstCV$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ResamplingFcstCV$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Tashman LJ (2000). "Out-of-sample tests of forecasting accuracy: an analysis and review." *International Journal of Forecasting*, **16**(4), 437–450.

Bergmeir C, Hyndman RJ, Koo B (2018). "A note on the validity of cross-validation for evaluating autoregressive time series prediction." *Computational Statistics & Data Analysis*, **120**, 70–83.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter3/evaluation_and_benchmarking.html#sec-resampling`
- Package **mlr3spatiotempcv** for spatio-temporal resamplings.
- Dictionary of Resamplings: mlr3::mlr_resamplings
- `as.data.table(mlr_resamplings)` for a table of available Resamplings in the running session (depending on the loaded packages).
- **mlr3spatiotempcv** for additional mlr3::Resamplings for spatio-temporal tasks.

Other Resampling: `mlr_resamplings_fcst.holdout`

Examples

```
# Create a task with 20 observations
task = tsk("airpassengers")
task$filter(1:20)

# Instantiate Resampling
cv = rsmp("fcst.cv", folds = 3, fixed_window = FALSE)
cv$instantiate(task)

# Individual sets:
cv$train_set(1)
cv$test_set(1)
intersect(cv$train_set(1), cv$test_set(1))

# Internal storage:
cv$instance #  list
```

`mlr_resamplings_fcst.holdout`
Forecast Holdout Resampling

Description

Splits data into a training set and a test set. Parameter `ratio` determines the ratio of observation going into the training set.

Dictionary

This Resampling can be instantiated via the dictionary mlr_resamplings or with the associated sugar function rsmp():

```
mlr_resamplings$get("fcst.holdout")
rsmp("fcst.holdout")
```

Parameters

- `ratio` (`numeric(1)`)
  Ratio of observations to put into the training set. Mutually exclusive with parameter `n`.
- `n` (`integer(1)`)
  Number of observations to put into the training set. If negative, the absolute value determines the number of observations in the test set. Mutually exclusive with parameter `ratio`.

Super class

`mlr3::Resampling` -> `ResamplingFcstHoldout`

Active bindings

`iters (integer(1))`
: Returns the number of resampling iterations, depending on the values stored in the `param_set`.

Methods

Public methods:

- `ResamplingFcstHoldout$new()`
- `ResamplingFcstHoldout$clone()`

`ResamplingFcstHoldout$new()`: Creates a new instance of this R6 class.

Usage:

```
ResamplingFcstHoldout$new()
```

`ResamplingFcstHoldout$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ResamplingFcstHoldout$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter3/evaluation_and_benchmarking.html#sec-resampling
- Package **mlr3spatiotempcv** for spatio-temporal resamplings.
- Dictionary of Resamplings: mlr3::mlr_resamplings
- `as.data.table(mlr_resamplings)` for a table of available Resamplings in the running session (depending on the loaded packages).
- **mlr3spatiotempcv** for additional mlr3::Resamplings for spatio-temporal tasks.

Other Resampling: `mlr_resamplings_fcst.cv`

Examples

```
# Create a task with 10 observations
task = tsk("airpassengers")
task$filter(1:10)

# Instantiate Resampling
holdout = rsmp("fcst.holdout", ratio = 0.5)
holdout$instantiate(task)

# Individual sets:
holdout$train_set(1)
holdout$test_set(1)

# Disjunct sets:
```

```
intersect(holdout$train_set(1), holdout$test_set(1))

# Internal storage:
holdout$instance # simple list
```

`mlr_tasks_airpassengers`

Air Passengers Forecast Task

Description

A forecast task for the popular datasets::AirPassengers data set. The task represents the monthly totals of international airline passengers from 1949 to 1960.

Format

R6::R6Class inheriting from TaskFcst.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function tsk():

```
mlr_tasks$get("airpassengers")
tsk("airpassengers")
```

Meta Information

- Task type: "fcst"
- Dimensions: 144x1
- Properties: "ordered"
- Has Missings: `FALSE`
- Target: "passengers"
- Features: -

Source

Box GEP, Jenkins GM (1976). *Time Series Analysis: Forecasting and Control*, Revised edition. Holden-Day, San Francisco.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Package **mlr3data** for more toy tasks.
- Package **mlr3oml** for downloading tasks from https://www.openml.org.
- Package **mlr3viz** for some generic visualizations.
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session (depending on the loaded packages).
- **mlr3fselect** and **mlr3filters** for feature selection and feature filtering.
- Extension packages for additional task types:
  - Unsupervised clustering: **mlr3cluster**
  - Probabilistic supervised regression and survival analysis: https://mlr3proba.mlr-org.com/.

Other Task: TaskFcst, mlr_tasks_electricity, mlr_tasks_livestock, mlr_tasks_lynx, mlr_tasks_usaccdeaths

---

mlr_tasks_electricity *Daily electricity demand for Victoria, Australia Forecast Task*

Description

A forecast task for the tsibbledata::vic_elec data set. The task represents a daily time series and is ordered by `date`.

Format

R6::R6Class inheriting from TaskFcst.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function tsk():

```
mlr_tasks$get("electricity")
tsk("electricity")
```

Meta Information

- Task type: "fcst"
- Dimensions: 1096x3
- Properties: "ordered"
- Has Missings: `FALSE`
- Target: "demand"
- Features: "holiday", "temperature"

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Package **mlr3data** for more toy tasks.
- Package **mlr3oml** for downloading tasks from https://www.openml.org.
- Package **mlr3viz** for some generic visualizations.
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session (depending on the loaded packages).
- **mlr3fselect** and **mlr3filters** for feature selection and feature filtering.
- Extension packages for additional task types:
  - Unsupervised clustering: **mlr3cluster**
  - Probabilistic supervised regression and survival analysis: https://mlr3proba.mlr-org.com/.

Other Task: TaskFcst, mlr_tasks_airpassengers, mlr_tasks_livestock, mlr_tasks_lynx, mlr_tasks_usaccdeaths

---

mlr_tasks_livestock *Australian Livestock Slaughter Forecast Task*

Description

A forecast task for the tsibbledata::aus_livestock data set. The task represents a monthly time series and is ordered by `month`.

Format

R6::R6Class inheriting from TaskFcst.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function tsk():

```
mlr_tasks$get("livestock")
tsk("livestock")
```

Meta Information

- Task type: "fcst"
- Dimensions: 29364x1
- Properties: "ordered", "keys"
- Has Missings: `FALSE`
- Target: "count"
- Features: -

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Package **mlr3data** for more toy tasks.
- Package **mlr3oml** for downloading tasks from https://www.openml.org.
- Package **mlr3viz** for some generic visualizations.
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session (depending on the loaded packages).
- **mlr3fselect** and **mlr3filters** for feature selection and feature filtering.
- Extension packages for additional task types:
  - Unsupervised clustering: **mlr3cluster**
  - Probabilistic supervised regression and survival analysis: https://mlr3proba.mlr-org.com/.

Other Task: TaskFcst, mlr_tasks_airpassengers, mlr_tasks_electricity, mlr_tasks_lynx, mlr_tasks_usaccdeaths

---

`mlr_tasks_lynx` *Annual Canadian Lynx Trappings Forecast Task*

Description

A forecast task for the popular datasets::lynx data set. The task represents the annual numbers of lynx trappings in Canada from 1821 to 1934.

Format

R6::R6Class inheriting from TaskFcst.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function tsk():

```
mlr_tasks$get("lynx")
tsk("lynx")
```

Meta Information

- Task type: "fcst"
- Dimensions: 114x1
- Properties: "ordered"
- Has Missings: FALSE
- Target: "count"
- Features: -

Source

Brockwell PJ, Davis RA (1991). *Time Series: Theory and Methods*, 2nd edition. Springer, New York.

References

Campbell MJ, Walker AM (1977). "A Survey of Statistical Work on the Mackenzie River Series of Annual Canadian Lynx Trappings for the Years 1821-1934 and a New Analysis." *Journal of the Royal Statistical Society. Series A (General)*, **140**(4), 411–431. doi:10.2307/2345277.

Becker RA, Chambers JM, Wilks AR (1988). *The New S Language*. Chapman and Hall/CRC, London.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Package **mlr3data** for more toy tasks.
- Package **mlr3oml** for downloading tasks from https://www.openml.org.
- Package **mlr3viz** for some generic visualizations.
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session (depending on the loaded packages).
- **mlr3fselect** and **mlr3filters** for feature selection and feature filtering.
- Extension packages for additional task types:
  - Unsupervised clustering: **mlr3cluster**
  - Probabilistic supervised regression and survival analysis: https://mlr3proba.mlr-org.com/.

Other Task: TaskFcst, mlr_tasks_airpassengers, mlr_tasks_electricity, mlr_tasks_livestock, mlr_tasks_usaccdeaths

---

mlr_tasks_usaccdeaths *Accidental Deaths in the US Forecast Task*

Description

A forecast task for the popular datasets::USAccDeaths data set. The task represents the monthly totals of accidental deaths in the US from 1973 to 1978.

Format

R6::R6Class inheriting from TaskFcst.

Dictionary

This Task can be instantiated via the dictionary mlr_tasks or with the associated sugar function tsk():

```
mlr_tasks$get("usaccdeaths")
tsk("usaccdeaths")
```

Meta Information

- Task type: "fcst"
- Dimensions: 72x1
- Properties: "ordered"
- Has Missings: FALSE
- Target: "deaths"
- Features: -

Source

Brockwell PJ, Davis RA (1991). *Time Series: Theory and Methods*, 2nd edition. Springer, New York.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Package **mlr3data** for more toy tasks.
- Package **mlr3oml** for downloading tasks from https://www.openml.org.
- Package **mlr3viz** for some generic visualizations.
- Dictionary of Tasks: mlr3::mlr_tasks
- as.data.table(mlr_tasks) for a table of available Tasks in the running session (depending on the loaded packages).
- **mlr3fselect** and **mlr3filters** for feature selection and feature filtering.
- Extension packages for additional task types:
  - Unsupervised clustering: **mlr3cluster**
  - Probabilistic supervised regression and survival analysis: https://mlr3proba.mlr-org.com/.

Other Task: TaskFcst, mlr_tasks_airpassengers, mlr_tasks_electricity, mlr_tasks_livestock, mlr_tasks_lynx

partition.TaskFcst *Manually Partition into Training, Test and Validation Set*

Description

Creates a split of the row ids of a Task into a training and a test set, and optionally a validation set.

Usage

```
## S3 method for class 'TaskFcst'
partition(task, ratio = 0.67)
```

Arguments

task	(Task) Task to operate on.
ratio	(numeric()) Ratio of observations to put into the training set. If a 2 element vector is provided, the first element is the ratio for the training set, the second element is the ratio for the test set. The validation set will contain the remaining observations.

Examples

```
task = tsk("airpassengers")
split = partition(task, ratio = 0.8)
```

PredictionFcst *Prediction Object for Forecasting*

Description

This object wraps the predictions returned by a forecast learner (LearnerFcst, RecursiveForecaster, DirectForecaster). It subclasses mlr3::PredictionRegr, so forecasting is treated as regression: the `response`, `se`, `quantiles` and `distr` fields and all regression measures continue to work.

In addition, the prediction carries the time index (and any key columns) of the forecast horizon in its `$data$extra` slot and their roles in `$col_roles`. These are exposed via the `$order` and `$key` fields, lead the `as.data.table()` output, and are used by `autoplot.PredictionFcst()` to draw a forecast plot.

The `task_type` is kept as `"regr"` so that regression measures remain compatible: forecasting is scored as regression.

Super classes

`mlr3::Prediction` -> `mlr3::PredictionRegr` -> `PredictionFcst`

Active bindings

`col_roles` (named `list()`)
: Column roles for `$data$extra`, with elements `order` and `key`.

`order` (`data.table::data.table()` | `NULL`)
: The forecast time index, recovered from `$data$extra`. A table with two columns:

- `row_id` (`integer()`), and
- `order` (`Date()` | `POSIXct()` | `integer()` | `numeric()`).

Returns `NULL` if no order role is stored.

`key` (`data.table::data.table()` | `NULL`)
: The series identity columns of the forecast horizon, recovered from `$data$extra`. A table with two or more columns:

- `row_id` (`integer()`), and
- key variable(s) (`character()` | `integer()` | `factor()` | `ordered()`).

If there is only one key column, it is named `key`. Returns `NULL` if there are no key columns.

Methods

Public methods:

- `PredictionFcst$new()`
- `PredictionFcst$clone()`

`PredictionFcst$new()`: Creates a new instance of this R6 class.

Usage:

```
PredictionFcst$new(
  task = NULL,
  row_ids = task$row_ids,
  truth = task$truth(),
  response = NULL,
  se = NULL,
  quantiles = NULL,
  distr = NULL,
  weights = NULL,
  check = TRUE,
  extra = NULL,
  col_roles = NULL,
  raw = NULL
)
```

Arguments:

`task` (TaskFcst)
: Task, used to extract defaults for `row_ids` and `truth`.

`row_ids` (`integer()`)
: Row ids of the predicted observations, i.e. the row ids of the test set.

`truth` (`numeric()`)
: True (observed) response.

`response` (`numeric()`)
: Vector of numeric response values. One element for each observation in the test set.

`se` (`numeric()`)
: Numeric vector of predicted standard errors. One element for each observation in the test set.

`quantiles` (`matrix()`)
: Numeric matrix of predicted quantiles. One row per observation, one column per quantile.

`distr` (`VectorDistribution`)
: `VectorDistribution` from package distr6 (in repository https://raphaels1.r-universe.dev).

`weights` (`numeric()`)
: Vector of measure weights for each observation.

`check` (`logical(1)`)
: If `TRUE`, performs some argument checks and predict type conversions.

`extra` (`list()`)
: Named list carrying the order (time) column and any key columns of the forecast horizon. The list names are the original task column names.

`col_roles` (named `list()`)
: Column roles for `extra`, with elements `order` and `key`. If `NULL`, the roles are derived from `task`; without a task, the extra columns carry no roles.

`raw` (any)
: Raw prediction object from the upstream model. Stored as-is without validation.

`PredictionFcst$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PredictionFcst$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Package **mlr3viz** for some generic visualizations.

Examples

```
task = tsk("airpassengers")
learner = lrn("fcst.auto_arima")$train(task)
p = forecast(learner, task, h = 12)
p$predict_types
head(as.data.table(p))
```

read_tsf *Read tsf files*

Description

Parses a file located at `file` and returns a `data.table::data.table()`.

Usage

```
read_tsf(file)
```

Arguments

`file`	`(character(1))` The path to the TSF file.

Value

(`data.table::data.table()`) with class `"tsf"`. If the file contains a frequency or horizon, the `"frequency"` and `"horizon"` attributes are set, respectively.

References

Godahewa R, Bergmeir C, Webb GI, Hyndman RJ, Montero-Manso P (2021). "Monash time series forecasting archive." *arXiv preprint arXiv:2105.06643*.

Examples

```
file = system.file("extdata", "m3_yearly_dataset.tsf", package = "mlr3forecast")
dt = read_tsf(file)
head(dt)
```

RecursiveForecaster *Recursive Forecast Learner*

Description

A mlr3::Learner for iterative one-step-ahead forecasting: a single model is fit, then applied recursively, feeding each prediction back as a lag/rolling feature for the next step.

Can be constructed in two ways:

- **Simple**: `RecursiveForecaster$new(learner, lags = 1:3)` – internally builds `po("fcst.lags", lags = lags) %>>% learner`.
- **Graph**: `RecursiveForecaster$new(graph)` – takes an arbitrary mlr3pipelines::Graph or mlr3pipelines::PipeOp.

Target transformations

A target transformation (e.g. mlr_pipeops_fcst.targetdiff, mlr3pipelines::PipeOpTargetMutate) must *wrap* the forecaster, not be placed *inside* its graph. Wrap it with mlr3pipelines::ppl("targettrafo") so the whole series is transformed once up front, the recursion runs entirely on the transformed scale, and predictions are inverted once at the end:

```
flrn = as_learner(ppl("targettrafo",
  graph = RecursiveForecaster$new(lrn("regr.rpart"), lags = 1:12),
  trafo_pipeop = po("fcst.targetdiff", lag = 1L)
))
flrn$train(task, split$train)
flrn$predict(task, split$test)  # predictions are on the original scale
```

Placing a mlr3pipelines::PipeOpTargetTrafo *inside* the graph is not supported and is rejected at construction.

Prediction uncertainty

Only the point forecast is fed back between steps, so `se`/`distr` uncertainty does not accumulate across horizons and intervals are too narrow for `h > 1`. For calibrated multi-step intervals, prefer DirectForecaster.

Validation and internal tuning

If the wrapped graph contains a learner with the `"validation"` property, the forecaster supports validation and internal tuning as well. Configure it with `mlr3::set_validate()`, which sets the forecaster's `$validate` field and routes the validation data to the graph's base learner. A numeric ratio is split chronologically per key (via `partition()`), so the validation set is always the most recent fraction of each series. Note that the validation scores measure one-step-ahead (teacher-forced) accuracy, not recursive multi-step accuracy.

Super class

`mlr3::Learner` -> `RecursiveForecaster`

Active bindings

`learner` (mlr3::Learner)
: The base regression learner.

`native_model` (any)
: The fitted model.

`lags` (`integer()` | `NULL`)
: The lags used, or `NULL` if no PipeOpFcstLags is in the graph.

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`marshaled` (`logical(1)`)
: Whether the learner's model is currently in marshaled form.

`validate` (`numeric(1)` | `"predefined"` | `"test"` | `NULL`)
How to construct the internal validation data. Use `mlr3::set_validate()` to also configure the wrapped graph.

`internal_valid_scores` (named `list()` | `NULL`)
The internal validation scores extracted from the wrapped graph, or `NULL` if no validation was done.

`internal_tuned_values` (named `list()` | `NULL`)
The internally tuned values extracted from the wrapped graph, or `NULL` if no internal tuning was done.

`predict_type` (`character(1)`)
Stores the currently active predict type.

Methods

Public methods:

- `RecursiveForecaster$new()`
- `RecursiveForecaster$print()`
- `RecursiveForecaster$marshal()`
- `RecursiveForecaster$unmarshal()`
- `RecursiveForecaster$importance()`
- `RecursiveForecaster$selected_features()`
- `RecursiveForecaster$oob_error()`
- `RecursiveForecaster$clone()`

`RecursiveForecaster$new()`: Creates a new instance of this R6 class.

Usage:

```
RecursiveForecaster$new(
  learner,
  lags = NULL,
  id = NULL,
  param_vals = list(),
  predict_type = NULL,
  clone_graph = TRUE
)
```

Arguments:

`learner` (mlr3::Learner | mlr3pipelines::Graph | mlr3pipelines::PipeOp)
A regression learner (when `lags` is provided) or a graph/PipeOp.

`lags` (`integer()` | `NULL`)
The lag values to use for creating lag features. If provided, `learner` is wrapped with `po("fcst.lags", lags = lags)`. If `NULL`, `learner` must be a mlr3pipelines::Graph or mlr3pipelines::PipeOp.

`id` (`character(1)` | `NULL`)
Identifier, default `NULL` (auto-generated).

`param_vals` (named `list()`)
List of hyperparameter settings.

predict_type (character(1) | NULL)
The predict type, default NULL.
clone_graph (logical(1))
Whether to clone the graph, default TRUE.

RecursiveForecaster$print(): Printer.

Usage:

```
RecursiveForecaster$print(...)
```

Arguments:

... (ignored).

RecursiveForecaster$marshal(): Marshal the learner's model.

Usage:

```
RecursiveForecaster$marshal(...)
```

Arguments:

... (any)
Additional arguments passed to mlr3::marshal_model().

RecursiveForecaster$unmarshal(): Unmarshal the learner's model.

Usage:

```
RecursiveForecaster$unmarshal(...)
```

Arguments:

... (any)
Additional arguments passed to mlr3::unmarshal_model().

RecursiveForecaster$importance(): The importance scores of the base learner, if it supports them.

Usage:

```
RecursiveForecaster$importance()
```

Returns: Named numeric().

RecursiveForecaster$selected_features(): The selected features of the base learner, if it supports them.

Usage:

```
RecursiveForecaster$selected_features()
```

Returns: character().

RecursiveForecaster$oob_error(): The out-of-bag error of the base learner, if it supports it.

Usage:

```
RecursiveForecaster$oob_error()
```

Returns: numeric(1).

RecursiveForecaster$clone(): The objects of this class are cloneable with this method.

Usage:

```
RecursiveForecaster$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
library(mlr3pipelines)

task = tsk("airpassengers")
flrn = RecursiveForecaster$new(lrn("regr.rpart"), lags = 1:3)
split = partition(task, ratio = 0.8)
flrn$train(task, split$train)
flrn$predict(task, split$test)

# graph: custom preprocessing pipeline
graph = po("fcst.lags", lags = 1:3) %>>% lrn("regr.rpart")
flrn = RecursiveForecaster$new(graph)
flrn$train(task, split$train)
flrn$predict(task, split$test)
```

---

recursive_forecaster *Create a Recursive Forecast Learner*

Description

Function to create a RecursiveForecaster object. This is the recommended way to construct a recursive forecaster. It is a thin wrapper around `RecursiveForecaster$new()`.

A recursive forecaster trains a single regression model and forecasts iteratively one step ahead, feeding each prediction back as a lag/rolling feature for the next step. For the direct strategy (one model per horizon) see `direct_forecaster()`.

Usage

```
recursive_forecaster(
  learner,
  lags = NULL,
  id = NULL,
  param_vals = list(),
  predict_type = NULL,
  clone_graph = TRUE
)
```

Arguments

`learner`	(mlr3::Learner | mlr3pipelines::Graph | mlr3pipelines::PipeOp) A regression learner (when `lags` is provided) or a graph/PipeOp.
`lags`	(`integer()` | `NULL`) The lag values to use for creating lag features. If provided, `learner` is wrapped with `po("fcst.lags", lags = lags)`. If `NULL`, `learner` must be a mlr3pipelines::Graph or mlr3pipelines::PipeOp.
`id`	(`character(1)` | `NULL`) Identifier, default `NULL` (auto-generated).

param_vals	(named list()) List of hyperparameter settings.
predict_type	(character(1) | NULL) The predict type, default NULL.
clone_graph	(logical(1)) Whether to clone the graph, default TRUE.

Value

RecursiveForecaster.

Examples

```
library(mlr3pipelines)

task = tsk("airpassengers")
split = partition(task, ratio = 0.8)

# simple: wrap a regression learner with lag features
flrn = recursive_forecaster(lrn("regr.rpart"), lags = 1:3)
flrn$train(task, split$train)
flrn$predict(task, split$test)

# graph: custom preprocessing pipeline
graph = po("fcst.lags", lags = 1:3) %>>% lrn("regr.rpart")
flrn = recursive_forecaster(graph)
```

---

selector_fcst_lags *Select Forecast Lag Features*

Description

mlr3pipelines::Selector that selects lag features created by PipeOpFcstLags. Matches features named {target}_lag_{i} where {target} is the task's target variable.

Usage

```
selector_fcst_lags()
```

Value

function: A mlr3pipelines::Selector function.

See Also

Other Selectors: selector_fcst_rolling()

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
pop = po("fcst.lags", lags = 1:3)
new_task = pop$train(list(task))[[1L]]
selector_fcst_lags()(new_task)
```

---

selector_fcst_rolling *Select Forecast Rolling Features*

Description

mlr3pipelines::Selector that selects rolling-window features created by PipeOpFcstRolling. Matches features named `{target}_roll_{fun}_{size}` where `{target}` is the task's target variable and `{fun}` is one of the aggregation functions supported by PipeOpFcstRolling.

Usage

```
selector_fcst_rolling()
```

Value

`function`: A mlr3pipelines::Selector function.

See Also

Other Selectors: `selector_fcst_lags()`

Examples

```
library(mlr3pipelines)
task = tsk("airpassengers")
pop = po("fcst.rolling", funs = c("mean", "sd"), window_sizes = c(3L, 12L))
new_task = pop$train(list(task))[[1L]]
selector_fcst_rolling()(new_task)
```

---

set_validate.RecursiveForecaster

Configure Validation for a RecursiveForecaster

Description

Sets the `$validate` field of the forecaster, which controls *how* the validation data is constructed (see mlr3::Learner), and configures the wrapped graph so its base learner uses it (via `mlr3pipelines::set_validate.GraphLear` the inner PipeOps receive `"predefined"`).

Usage

```
## S3 method for class 'RecursiveForecaster'
set_validate(
  learner,
  validate,
  ids = NULL,
  args_all = list(),
  args = list(),
  ...
)
```

Arguments

learner	(RecursiveForecaster) The forecaster to configure.
validate	(numeric(1) | "predefined" | "test" | NULL) How to construct the internal validation data.
ids	(character() | NULL) The ids of the PipeOps for which to enable validation, forwarded to mlr3pipelines::set_validate.Gra Defaults to the base learner.
args_all	(named list()) Arguments passed to all set_validate() calls of the affected PipeOps.
args	(named list() of named list()s) Arguments passed to the set_validate() calls of specific PipeOps, named by their ids.
...	(any) Further arguments passed to mlr3pipelines::set_validate.GraphLearner().

Value

RecursiveForecaster, invisibly.

TaskFcst *Forecast Task*

Description

This task specializes mlr3::Task, mlr3::TaskSupervised and mlr3::TaskRegr for forecasting problems. The target column is assumed to be numeric. The task_type is set to "fcst".

It is recommended to use as_task_fcst() for construction. Predefined tasks are stored in the dictionary mlr3::mlr_tasks.

Series identity and keys

A task may have one or more `key` columns whose combination identifies each series. The key role drives all per-series operations: target lags and rolling windows (PipeOpFcstLags, PipeOpFcstRolling) are computed within each series, and the future forecast grid is built per series.

Key columns are **not** features by default. To let a global model specialize per series, add the feature role back:

```
task$set_col_roles("series_id", add_to = "feature")
```

For a high-cardinality key, encode it inside the learner graph (e.g. `po("encodeimpact")` or `po("encodelmer")`) rather than passing the raw factor to the model. Note that keys are labels regardless of their type: an integer key used as a feature is passed to the learner as a number, which is rarely intended – convert it to a factor first.

Super classes

`mlr3::Task` -> `mlr3::TaskSupervised` -> `mlr3::TaskRegr` -> `TaskFcst`

Active bindings

`freq` (`character(1)` | `numeric(1)` | `NULL`)
The frequency of the time series.

`properties` (`character()`)
Set of task properties. Possible properties are stored in mlr_reflections$task_properties. The following properties are currently standardized and understood by tasks in **mlr3**:

- `"strata"`: The task is resampled using one or more stratification variables (role `"stratum"`).
- `"groups"`: The task comes with grouping/blocking information (role `"group"`).
- `"weights_learner"`: The task comes with observation weights for the learner (role `"weights_learner"`).
- `"weights_measure"`: The task comes with observation weights for the measure (role `"weights_measure"`).
- `"offset"`: The task comes with offset information (role `"offset"`).
- `"ordered"`: The task has columns which define the row order (role `"order"`).
- `"keys"`: The task has columns which define the time series `"key"`.

Note that above listed properties are calculated from the `$col_roles` and may not be set explicitly.

`order` (`data.table::data.table()`)
A table with two columns:

- `row_id` (`integer()`), and
- `order` (`Date()` | `POSIXct()` | `integer()` | `numeric()`).

`key` (`data.table::data.table()` | `NULL`)
If the task has a column with designated role `"key"`, a table with two or more columns:

- `row_id` (`integer()`), and
- key variable(s) (`character()` | `integer()` | `factor()` | `ordered()`).

If there is only one key column, it will be named as `key`. Returns `NULL` if there are no key columns.

Methods

Public methods:

- TaskFcst$new()
- TaskFcst$view()
- TaskFcst$print()
- TaskFcst$clone()

`TaskFcst$new()`: Creates a new instance of this R6 class. The function `as_task_fcst()` provides an alternative way to construct forecast tasks.

Usage:

```
TaskFcst$new(
  id,
  backend,
  target,
  order,
  key = character(),
  freq = NULL,
  label = NA_character_,
  extra_args = list()
)
```

Arguments:

`id` (`character(1)`)
: Identifier for the new instance.

`backend` (mlr3::DataBackend)
: Either a mlr3::DataBackend, or any object which is convertible to a mlr3::DataBackend with `as_data_backend()`. E.g., a `data.frame()` will be converted to a mlr3::DataBackendDataTable.

`target` (`character(1)`)
: Name of the target column.

`order` (`character(1)`)
: Name of the order column.

`key` (`character()`)
: Names of the columns whose combination identifies a series. Key columns must be character, integer, factor, or ordered columns.

`freq` (`character(1)` | `numeric(1)` | `NULL`)
: Frequency of the time series. A `seq()`-compatible string gives the calendar step of the time grid, e.g. `"1 month"`, `"day"`, `"3 months"`, `"1 hour"`, `"week"`. A positive number gives the seasonal period (as in `stats::ts()`) for an integer or numeric order column; the grid step is then inferred from the data.

`label` (`character(1)`)
: Label for the new instance.

`extra_args` (named `list()`)
: Named list of constructor arguments, required for converting task types via `mlr3::convert_task()`.

`TaskFcst$view()`: Returns a slice of the data from the mlr3::DataBackend as a `data.table::data.table()`. Rows default to observations with role `"use"`, and columns default to features with roles `"target"`,

"order", "key" or "feature". If rows or cols are specified which do not exist in the mlr3::DataBackend, an exception is raised.

Rows and columns are returned in the order specified via the arguments rows and cols. If rows is NULL, rows are returned in the order of task$row_ids. If cols is NULL, the column order defaults to c(task$target_names, task$feature_names, taskcol_roleskey, taskcol_rolesorder). Note that it is recommended to **not** rely on the order of columns, and instead always address columns with their respective column name.

Usage:

```
TaskFcst$view(rows = NULL, cols = NULL, ordered = FALSE)
```

Arguments:

`rows` (positive `integer()` | `NULL`)
Vector or row indices.

`cols` (`character()` | `NULL`)
Vector of column names.

`ordered` (`logical(1)`)
If `TRUE`, data is ordered according to the columns with column role "order" and "key".

Returns: A `data.table::data.table()`.

`TaskFcst$print()`: Printer.

Usage:

```
TaskFcst$print(...)
```

Arguments:

`...` (ignored).

`TaskFcst$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TaskFcst$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html
- Package **mlr3data** for more toy tasks.
- Package **mlr3oml** for downloading tasks from https://www.openml.org.
- Package **mlr3viz** for some generic visualizations.
- Dictionary of Tasks: mlr3::mlr_tasks
- `as.data.table(mlr_tasks)` for a table of available Tasks in the running session (depending on the loaded packages).
- **mlr3fselect** and **mlr3filters** for feature selection and feature filtering.
- Extension packages for additional task types:

- Unsupervised clustering: **mlr3cluster**
- Probabilistic supervised regression and survival analysis: https://mlr3proba.mlr-org.com/.

Other Task: mlr_tasks_airpassengers, mlr_tasks_electricity, mlr_tasks_livestock, mlr_tasks_lynx, mlr_tasks_usaccdeaths

Examples

```
library(data.table)
airpassengers = tsbox::ts_dt(AirPassengers)
setnames(airpassengers, c("month", "passengers"))
task = as_task_fcst(airpassengers, target = "passengers", order = "month", freq = "month")
task$task_type
task$formula()
task$truth()
```

Index