Package ‘ngboostForecast’

August 6, 2022

Title Probabilistic Time Series Forecasting

Version 0.1.1

Description
Probabilistic time series forecasting via Natural Gradient Boosting for Probabilistic Prediction.

License Apache License (>= 2)

URL https://github.com/Akai01/ngboostForecast

BugReports https://github.com/Akai01/ngboostForecast/issues

Encoding UTF-8

LazyData true

SystemRequirements Python (>= 3.6)

RoxygenNote 7.2.0

Imports dplyr (>= 1.0.7), forecast (>= 8.15), magrittr (>= 2.0.1), R6 (>= 2.5.1)

Suggests ggplot2 (>= 3.3.5), testthat (>= 3.0.0)

Config/testthat/edition 3

Config/reticulate list( packages = list( list(package = 'importlib-metadata', pip = TRUE), list(package = 'ngboost', pip = TRUE) ) )

Depends R (>= 3.6), reticulate (>= 1.20)

NeedsCompilation no

Author Resul Akay [aut, cre]

Maintainer Resul Akay <resulakay1@gmail.com>

Repository CRAN

Date/Publication 2022-08-06 11:30:08 UTC
Description

NGBoost distributions

Usage

Dist(
  dist = c("Normal", "Bernoulli", "k_categorical", "StudentT", "Laplace", "Cauchy",
           "Exponential", "LogNormal", "MultivariateNormal", "Poisson"),
  k
)

Arguments

dist  NGBoost distributions. One of the following:
  • Bernoulli
  • k_categorical
  • StudentT
  • Poisson
  • Laplace
  • Cauchy
  • Exponential
  • LogNormal
  • MultivariateNormal
  • Normal

k  Used only with k_categorical and MultivariateNormal

Value

An NGBoost Distribution object
is_exists_conda

Is conda installed?

Description

Only for internal usage.

Usage

is_exists_conda()

Value

Logical, TRUE if conda is installed.

Author(s)

Resul Akay

NGBforecast

NGBoost forecasting class

Description

The main forecasting class.

Value

An NGBforecast class

Methods

Public methods:

- NGBforecast$new()
- NGBforecast$fit()
- NGBforecast$forecast()
- NGBforecast$feature_importances()
- NGBforecast$plot_feature_importance()
- NGBforecast$get_params()
- NGBforecast$clone()

Method new(): Initialize an NGBforecast model.

Usage:
NGBforecast$new(
  Dist = NULL,
  Score = NULL,
  Base = NULL,
  natural_gradient = TRUE,
  n_estimators = as.integer(500),
  learning_rate = 0.01,
  minibatch_frac = 1,
  col_sample = 1,
  verbose = TRUE,
  verbose_eval = as.integer(100),
  tol = 1e-04,
  random_state = NULL
)

Arguments:
Dist Assumed distributional form of Y|X=x. An output of Dist function, e.g. Dist('Normal')
Score Rule to compare probabilistic predictions to the observed data. A score from Scores function, e.g. Scores(score = "LogScore").
Base Base learner. An output of sklearner function, e.g. sklearner(module = "tree",
                           class = "DecisionTreeRegressor", ...)
natural_gradient Logical flag indicating whether the natural gradient should be used
n_estimators The number of boosting iterations to fit
learning_rate The learning rate
minibatch_frac The percent subsample of rows to use in each boosting iteration
col_sample The percent subsample of columns to use in each boosting iteration
verbose Flag indicating whether output should be printed during fitting. If TRUE it will print logs.
verbose_eval Increment (in boosting iterations) at which output should be printed
tol Numerical tolerance to be used in optimization
random_state Seed for reproducibility.

Returns: An NGBforecast object that can be fit.

Method fit(): Fit the initialized model.

Usage:
NGBforecast$fit(
  y,
  max_lag = 5,
  xreg = NULL,
  test_size = NULL,
  seasonal = TRUE,
  K = frequency(y)/2 - 1,
  train_loss_monitor = NULL,
  val_loss_monitor = NULL,
  early_stopping_rounds = NULL
)
Arguments:
y A time series (ts) object
max_lag Maximum number of lags
xreg Optional. A numerical matrix of external regressors, which must have the same number of rows as y.
test_size The length of validation set. If it is NULL, then, it is automatically specified.
seasonal Boolean. If seasonal = TRUE the fourier terms will be used for modeling seasonality.
K Maximum order(s) of Fourier terms, used only if seasonal = TRUE.
train_loss_monitor A custom score or set of scores to track on the training set during training. Defaults to the score defined in the NGBoost constructor. Please do not modify unless you know what you are doing.
val_loss_monitor A custom score or set of scores to track on the validation set during training. Defaults to the score defined in the NGBoost constructor. Please do not modify unless you know what you are doing.
early_stopping_rounds The number of consecutive boosting iterations during which the loss has to increase before the algorithm stops early.

Returns: NULL

Method forecast(): Forecast the fitted model
Usage:
NGBforecast$forecast(h = 6, xreg = NULL, level = c(80, 95), data_frame = FALSE)
Arguments:
h Forecast horizon
xreg A numerical vector or matrix of external regressors
level Confidence level for prediction intervals
data_frame Bool. If TRUE, forecast will be returned as a data.frame object, if FALSE it will return a forecast class. If TRUE, autoplot will function.

Method feature_importances(): Return the feature importance for all parameters in the distribution (the higher, the more important the feature).
Usage:
NGBforecast$feature_importances()
Returns: A data frame

Method plot_feature_importance(): Plot feature importance
Usage:
NGBforecast$plot_feature_importance()
Returns: A ggplot object

Method get_params(): Get parameters for this estimator.
Usage:
NGBforecast$get_params(deep = TRUE)
Arguments:
deep bool, default = TRUE If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns: A named list of parameters.

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

Usage:
NGBforecast$clone(deep = FALSE)

Arguments:
dee Whether to make a deep clone.

Author(s)
Resul Akay

References
Duan, T et. al. (2019), NGBoost: Natural Gradient Boosting for Probabilistic Prediction.

Examples
## Not run:
library(ngboostForecast)

model <- NGBforecast$new(Dist = Dist("Normal"),
                         Base = sklearn(module = "linear_model",
                                       class = "Ridge"),
                         Score = Scores("LogScore"),
                         natural_gradient = TRUE,
                         n_estimators = 200,
                         learning_rate = 0.1,
                         minibatch_frac = 1,
                         col_sample = 1,
                         verbose = TRUE,
                         verbose_eval = 100,
                         tol = 1e-5)
model$fit(y = AirPassengers, seasonal = TRUE, max_lag = 12, xreg = NULL,
          early_stopping_rounds = 10L)
f <- model$forecast(h = 12, level = c(90, 80), xreg = NULL)

autoplot(f)

## End(Not run)
**NGBoost forecasting model selection class**

**Description**

It is a wrapper for the sklearn GridSearchCV with TimeSeriesSplit.

**Methods**

**Public methods:**

- `NGBforecastCV$new()`
- `NGBforecastCV$tune()`
- `NGBforecastCV$clone()`

**Method** `new()`: Initialize an NGBforecastCV model.

**Usage:**

```r
NGBforecastCV$new(
  Dist = NULL,
  Score = NULL,
  Base = NULL,
  natural_gradient = TRUE,
  n_estimators = as.integer(500),
  learning_rate = 0.01,
  minibatch_frac = 1,
  col_sample = 1,
  verbose = TRUE,
  verbose_eval = as.integer(100),
  tol = 1e-04,
  random_state = NULL
)
```

**Arguments:**

- `Dist`: Assumed distributional form of \(Y|X=x\). An output of `Dist` function, e.g. `Dist('Normal')`.
- `Score`: Rule to compare probabilistic predictions to the observed data. A score from `Scores` function, e.g. `Scores(score = "LogScore")`.
- `Base`: Base learner. An output of `sklearn` function, e.g. `sklearn(module = "tree", class = "DecisionTreeRegressor", ...)`.
- `natural_gradient`: Logical flag indicating whether the natural gradient should be used.
- `n_estimators`: The number of boosting iterations to fit.
- `learning_rate`: The learning rate.
- `minibatch_frac`: The percent subsample of rows to use in each boosting iteration.
- `col_sample`: The percent subsample of columns to use in each boosting iteration.
- `verbose`: Flag indicating whether output should be printed during fitting. If TRUE it will print logs.
- `verbose_eval`: Increment (in boosting iterations) at which output should be printed.
tol  Numerical tolerance to be used in optimization
random_state  Seed for reproducibility.

Returns:  An NGBforecastCV object that can be fit.

Method  tune(): Tune ngboosForecast.

Usage:
NGBforecastCV$tune(
y,  
max_lag = 5,
xreg = NULL,
seasonal = TRUE,
K = frequency(y)/2 - 1,
n_splits = NULL,
train_loss_monitor = NULL,
val_loss_monitor = NULL,
early_stopping_rounds = NULL
)

Arguments:
y  A time series (ts) object
max_lag  Maximum number of lags
xreg  Optional. A numerical matrix of external regressors, which must have the same number of rows as y.
seasonal  Boolean. If seasonal = TRUE the fourier terms will be used for modeling seasonality.
K  Maximum order(s) of Fourier terms, used only if seasonal = TRUE.
n_splits  Number of splits. Must be at least 2.
train_loss_monitor  A custom score or set of scores to track on the training set during training. Defaults to the score defined in the NGBoost constructor. Please do not modify unless you know what you are doing.
val_loss_monitor  A custom score or set of scores to track on the validation set during training. Defaults to the score defined in the NGBoost constructor. Please do not modify unless you know what you are doing.
early_stopping_rounds  The number of consecutive boosting iterations during which the loss has to increase before the algorithm stops early.
test_size  The length of validation set. If it is NULL, then, it is automatically specified.

Returns:  A named list of best parameters.

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

Usage:
NGBforecastCV$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

Author(s)
Resul Akay
Examples

```r
## Not run:

library(ngboostForecast)

dists <- list(Dist("Normal"))

base_learners <- list(sklearner(module = "tree", class = "DecisionTreeRegressor", max_depth = 1),
                        sklearner(module = "tree", class = "DecisionTreeRegressor", max_depth = 2),
                        sklearner(module = "tree", class = "DecisionTreeRegressor", max_depth = 3),
                        sklearner(module = "tree", class = "DecisionTreeRegressor", max_depth = 4),
                        sklearner(module = "tree", class = "DecisionTreeRegressor", max_depth = 5),
                        sklearner(module = "tree", class = "DecisionTreeRegressor", max_depth = 6),
                        sklearner(module = "tree", class = "DecisionTreeRegressor", max_depth = 7))

scores <- list(Scores("LogScore"))

model <- NGBforecastCV$new(Dist = dists,
                             Base = base_learners,
                             Score = scores,
                             natural_gradient = TRUE,
                             n_estimators = list(10, 100),
                             learning_rate = list(0.1, 0.2),
                             minibatch_frac = list(0.1, 1),
                             col_sample = list(0.3),
                             verbose = FALSE,
                             verbose_eval = 100,
                             tol = 1e-5)

params <- model$tune(y = AirPassengers,
                      seasonal = TRUE,
                      max_lag = 12,
                      xreg = NULL,
                      early_stopping_rounds = NULL,
                      n_splits = 4L)

params

## End(Not run)
```
Probabilistic time series forecasting via Natural Gradient Boosting for Probabilistic Prediction.

References

Duan, T et. al. (2019), NGBoost: Natural Gradient Boosting for Probabilistic Prediction.

Examples

```r
## Not run:
library(ngboostForecast)

model <- NGBforecast$new(Dist = Dist("Normal"),
  Base = sklearner(module = "linear_model",
  class = "Ridge"),
  Score = Scores("LogScore"),
  natural_gradient = TRUE,
  n_estimators = 200,
  learning_rate = 0.1,
  minibatch_frac = 1,
  col_sample = 1,
  verbose = TRUE,
  verbose_eval = 100,
  tol = 1e-5)
model$fit(y = AirPassengers, seasonal = TRUE, max_lag = 12, xreg = NULL,
  early_stopping_rounds = 10L)
fc <- model$forecast(h = 12, level = c(90, 80), xreg = NULL)
autoplot(fc)
## End(Not run)
```

Scores

Select a rule to compare probabilistic predictions to the observed data.

Description

Select a rule to compare probabilistic predictions to the observed data. A score from ngboost.scores, e.g. LogScore.
Usage

Scores(score = c("LogScore", "CRPS", "CRPScore", "MLE"))

Arguments

score A string. can be one of the following:
- LogScore : Generic class for the log scoring rule.
- CRPS : Generic class for the continuous ranked probability scoring rule.
- CRPScore : Generic class for the continuous ranked probability scoring rule.
- MLE : Generic class for the log scoring rule.

Value

A score class from ngboost.scores

Author(s)

Resul Akay

Description

The Seatbelts dataset from the datasets package.

Usage

seatbelts

Format

An object of class mts (inherits from ts) with 192 rows and 8 columns.

Source


References

**Description**

Scikit-Learn interface

**Usage**

```r
sklearner(module = "tree", class = "DecisionTreeRegressor", ...)
```

**Arguments**

- `module` : scikit-learn module name, default is 'tree'.
- `class` : scikit-learn's module class, default is 'DecisionTreeRegressor'
- `...` : Other arguments passed to model class

**Author(s)**

Resul Akay

**Examples**

```r
## Not run:

sklearner(module = "tree", class = "DecisionTreeRegressor", 
criterion="friedman_mse", min_samples_split=2)

## End(Not run)
```
Index

* datasets
  seatbelts, 11

autoplot, 5

Dist, 2, 4, 7

is_exists_conda, 3

NGBforecast, 3
NGBforecastCV, 7
ngboostForecast, 10

Scores, 4, 7, 10
seatbelts, 11
sklearner, 4, 7, 12