Package: crossvalidation (via r-universe)

September 26, 2024

Type Package

Title Generic cross-validation functions

Version 0.5.0

Date 2023-08-23

Author T. Moudiki

Maintainer T. Moudiki <thierry.moudiki@gmail.com>

Description Generic functions for cross-validation of
Statistical/Machine Learning models

License file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.2

BugReports https://github.com/Techtonique/crossvalidation/issues

Depends doSNOW, foreach, scoringRules

Suggests glmnet, testthat, forecast, randomForest, knitr, rmarkdown,
xgboost

VignetteBuilder knitr

Repository https://techtonique.r-universe.dev

RemoteUrl https://github.com/Techtonique/crossvalidation

RemoteRef HEAD

RemoteSha 727f217ac7482d35fd74c8a2aec28623622fbd7f

Contents

`boxplot.cvsamples` *Boxplots of cross-validation performances*

Description

Boxplots of cross-validation performances

Usage

```
## S3 method for class 'cvsamples'
boxplot(x, ...)
```

Arguments

`x`	a list containing models cross-validation performances, using `crossvalidation::create_samples`
`...`	additional parameters to be passed to `boxplot`

Examples

```
## Not run:
print("see vignettes")

## End(Not run)
```

`create_samples` *Create a data structure of cross-validation results*

Description

Create a data structure of cross-validation results

Usage

```
create_samples(..., model_names)
```

Arguments

`...`	list of cross-validation results for multiple models
`model_names`	model names

Value

a list of results to be used in `plot`

Examples

```
## Not run:
print("see vignettes")

## End(Not run)
```

`crossval_ml` *Generic cross-validation function*

Description

Generic cross-validation

Usage

```
crossval_ml(
  x,
  y,
  fit_func = crossvalidation::fit_lm,
  predict_func = crossvalidation::predict_lm,
  fit_params = NULL,
  k = 5,
  repeats = 3,
  p = 1,
  seed = 123,
  eval_metric = NULL,
  cl = NULL,
  errorhandling = c("stop", "remove", "pass"),
  packages = c("stats", "Rcpp"),
  verbose = FALSE,
  show_progress = TRUE,
  ...
)
```

Arguments

`x` input covariates' matrix

`y` response variable; a vector

`fit_func` a function for fitting the model

`predict_func` a function for predicting values from the model

`fit_params` a list; additional (model-specific) parameters to be passed to `fit_func` `k` an integer; number of folds in k-fold cross validation

`repeats` an integer; number of repeats for the k-fold cross validation

`p` a float; proportion of data in the training/testing set, default is 1 and must be > 0.5. If `p` < 1, a validation set error is calculated on the remaining 1-`p` fraction data

`seed` random seed for reproducibility of results

`eval_metric` a function measuring the test errors; if not provided: RMSE for regression and accuracy for classification

`cl` an integer; the number of clusters for parallel execution

`errorhandling` specifies how a task evalution error should be handled. If value is "stop", then execution will be stopped if an error occurs. If value is "remove", the result for that task will not be returned. If value is "pass", then the error object generated by task evaluation will be included with the rest of the results. The default value is "stop".

`packages` character vector of packages that the tasks depend on

`verbose` logical flag enabling verbose messages. This can be very useful for troubleshooting.

`show_progress` show evolution of the algorithm

`...` additional parameters

Examples

```
# dataset

set.seed(123)
n <- 1000 ; p <- 10
X <- matrix(rnorm(n * p), n, p)
y <- rnorm(n)

# linear model example -----

crossvalidation::crossval_ml(x = X, y = y,
                             k = 5L, repeats = 3L)


# randomForest example -----

require(randomForest)

# fit randomForest with mtry = 2

## Not run:
crossvalidation::crossval_ml(x = X, y = y, k = 5L, repeats = 3L,
                  fit_func = randomForest::randomForest, predict_func = predict,
                  packages = "randomForest", fit_params = list(mtry = 2))

# fit randomForest with mtry = 4
```

```
crossvalidation::crossval_ml(x = X, y = y, k = 5L, repeats = 3L,
                 fit_func = randomForest::randomForest, predict_func = predict,
                 packages = "randomForest", fit_params = list(mtry = 4))

fit randomForest with mtry = 4, with a validation set

crossvalidation::crossval_ml(x = X, y = y, k = 5, repeats = 2, p = 0.8,
                 fit_func = randomForest::randomForest, predict_func = predict,
                 packages = "randomForest", fit_params = list(mtry = 4))


## End(Not run)
```

crossval_ts *Generic cross-validation function for time series*

Description

Generic cross-validation for univariate and multivariate time series

Usage

```
crossval_ts(
  y,
  x = NULL,
  fit_func = crossvalidation::fit_lm,
  predict_func = crossvalidation::predict_lm,
  fcast_func = NULL,
  fit_params = NULL,
  p = 1,
  initial_window = 5,
  horizon = 3,
  fixed_window = TRUE,
  level = c(80, 95),
  seed = 123,
  eval_metric = NULL,
  cl = NULL,
  errorhandling = c("stop", "remove", "pass"),
  packages = c("stats", "Rcpp"),
  verbose = FALSE,
  show_progress = TRUE,
  ...
)
```

Arguments

`y` response time series; a vector or a matrix

`x` input covariates' matrix (optional) for ML models

`fit_func` a function for fitting the model (if validation of ML model)

`predict_func` a function for predicting values from the model (if validation of ML model)

`fcast_func` time series forecasting function (e.g forecast::thetaf)

`fit_params` a list; additional (model-specific) parameters to be passed to `fit_func`

`p` a float; percentage of original data in the training/testing procedure, default is 1 and must be > 0.5.

`initial_window` an integer; the initial number of consecutive values in each training set sample

`horizon` an integer; the number of consecutive values in test set sample

`fixed_window` a boolean; if FALSE, all training samples start at 1

`level` a numeric vector; confidence levels for prediction intervals.

`seed` random seed for reproducibility of results

`eval_metric` a function measuring the test errors; if not provided: RMSE for regression and accuracy for classification

`cl` an integer; the number of clusters for parallel execution

`errorhandling` specifies how a task evalution error should be handled. If value is "stop", then execution will be stopped if an error occurs. If value is "remove", the result for that task will not be returned. If value is "pass", then the error object generated by task evaluation will be included with the rest of the results. The default value is "stop".

`packages` character vector of packages that the tasks depend on

`verbose` logical flag enabling verbose messages. This can be very useful for troubleshooting.

`show_progress` show evolution of the algorithm

`...` additional parameters

Examples

```
require(forecast)
data("AirPassengers")

# Example 1 -----

res <- crossval_ts(y=AirPassengers, initial_window = 10,
horizon = 3, fcast_func = forecast::thetaf)
print(colMeans(res))


# Example 2 -----

## Not run:
```

```
fcast_func <- function (y, h, ...)
{
      forecast::forecast(forecast::auto.arima(y, ...),
      h=h, ...)
}

res <- crossval_ts(y=AirPassengers, initial_window = 10, horizon = 3,
fcast_func = fcast_func)
print(colMeans(res))

## End(Not run)


# Example 3 -----

fcast_func <- function (y, h, ...)
{
      forecast::forecast(forecast::ets(y, ...),
      h=h, ...)
}

res <- crossval_ts(y=AirPassengers,
initial_window = 10, horizon = 3, fcast_func = fcast_func)
print(colMeans(res))


# Example 4 -----

xreg <- cbind(1, 1:length(AirPassengers))
res <- crossval_ts(y=AirPassengers, x=xreg, fit_func = crossvalidation::fit_lm,
predict_func = crossvalidation::predict_lm,
initial_window = 10,
horizon = 3,
fixed_window = TRUE)
print(colMeans(res))


# Example 5 -----

res <- crossval_ts(y=AirPassengers, fcast_func = forecast::thetaf,
initial_window = 10,
horizon = 3,
fixed_window = TRUE)
print(colMeans(res))


#' # Example 6 -----

xreg <- cbind(1, 1:length(AirPassengers))
res <- crossval_ts(y=AirPassengers, x=xreg, fit_func = crossvalidation::fit_lm,
predict_func = crossvalidation::predict_lm,
initial_window = 10,
horizon = 3,
fixed_window = TRUE)
print(colMeans(res))


# Example 7 -----

x <- ts(matrix(rnorm(50), nrow = 25))

fcast_func <- function(y, h = 5, type_forecast=c("mean", "median"))
{
 type_forecast <- match.arg(type_forecast)

 if (type_forecast == "mean")
 {
  means <- colMeans(y)
  return(list(mean = t(replicate(n = h, expr = means))))
 } else {
  medians <- apply(y, 2, median)
  return(list(mean = t(replicate(n = h, expr = medians))))
 }

}

print(fcast_func(x))

res <- crossval_ts(y = x, fcast_func = fcast_func, fit_params = list(type_forecast = "median"))
colMeans(res)

res <- crossval_ts(y = x, fcast_func = fcast_func, fit_params = list(type_forecast = "mean"))
colMeans(res)

# Example 8 -----

eval_metric <- function(predicted, observed)
{
  error <- observed - predicted

  res <- apply(error, 2, function(x) sqrt(mean(x ^ 2, na.rm = FALSE)))

  return(res)
}

res <- crossval_ts(y = x, fcast_func = fcast_func, fit_params = list(type_forecast = "mean"),
eval_metric = eval_metric)

colMeans(res)
```

eval_ts *Rolling origin evaluation on validation set (time series)*

Description

Rolling origin evaluation on validation set (time series)

Usage

```
eval_ts(
  y,
  x = NULL,
  fit_func = crossvalidation::fit_lm,
  predict_func = crossvalidation::predict_lm,
  fcast_func = NULL,
  fit_params = NULL,
  q = 0.2,
  initial_window = 5,
  horizon = 3,
  fixed_window = TRUE,
  level = c(80, 95),
  seed = 123,
  eval_metric = NULL,
  cl = NULL,
  errorhandling = c("stop", "remove", "pass"),
  packages = c("stats", "Rcpp"),
  verbose = FALSE,
  show_progress = TRUE,
  ...
)
```

Arguments

`y`	response time series; a vector or a matrix
`x`	input covariates' matrix (optional) for ML models
`fit_func`	a function for fitting the model (if validation of ML model)
`predict_func`	a function for predicting values from the model (if validation of ML model)
`fcast_func`	time series forecasting function (e.g forecast::thetaf)
`fit_params`	a list; additional (model-specific) parameters to be passed to `fit_func`
`q`	a float; percentage of original data in the validation test.
`initial_window`	an integer; the initial number of consecutive values in each training set sample
`horizon`	an integer; the number of consecutive values in test set sample
`fixed_window`	a boolean; if FALSE, all training samples start at 1
`level`	a numeric vector; confidence levels for prediction intervals.
`seed`	random seed for reproducibility of results
`eval_metric`	a function measuring the test errors; if not provided: RMSE for regression and accuracy for classification
`cl`	an integer; the number of clusters for parallel execution

`errorhandling` specifies how a task evalution error should be handled. If value is "stop", then execution will be stopped if an error occurs. If value is "remove", the result for that task will not be returned. If value is "pass", then the error object generated by task evaluation will be included with the rest of the results. The default value is "stop".

`packages` character vector of packages that the tasks depend on

`verbose` logical flag enabling verbose messages. This can be very useful for troubleshooting.

`show_progress` show evolution of the algorithm

`...` additional parameters

Examples

```
require(forecast)
data("AirPassengers")

# Example 1 -----

res <- eval_ts(y=AirPassengers, initial_window = 10,
horizon = 3, fcast_func = forecast::thetaf)
print(colMeans(res))


# Example 2 -----

## Not run:
fcast_func <- function (y, h, ...)
{
      forecast::forecast(forecast::auto.arima(y, ...),
      h=h, ...)
}

res <- eval_ts(y=AirPassengers, initial_window = 10, horizon = 3,
fcast_func = fcast_func)
print(colMeans(res))

## End(Not run)


# Example 3 -----

fcast_func <- function (y, h, ...)
{
      forecast::forecast(forecast::ets(y, ...),
      h=h, ...)
}

res <- eval_ts(y=AirPassengers,
initial_window = 10, horizon = 3, fcast_func = fcast_func)
print(colMeans(res))
```

```
# Example 4 -----

xreg <- cbind(1, 1:length(AirPassengers))
res <- eval_ts(y=AirPassengers, x=xreg,
fit_func = crossvalidation::fit_lm,
predict_func = crossvalidation::predict_lm,
initial_window = 10,
horizon = 3,
fixed_window = TRUE)
print(colMeans(res))


# Example 5 -----

res <- eval_ts(y=AirPassengers, fcast_func = forecast::thetaf,
initial_window = 10,
horizon = 3,
fixed_window = TRUE)
print(colMeans(res))


#' # Example 6 -----

xreg <- cbind(1, 1:length(AirPassengers))
res <- eval_ts(y=AirPassengers, x=xreg,
fit_func = crossvalidation::fit_lm,
predict_func = crossvalidation::predict_lm,
initial_window = 10,
horizon = 3,
fixed_window = TRUE)
print(colMeans(res))


# Example 7 -----

x <- ts(matrix(rnorm(50), nrow = 25))

fcast_func <- function(y, h = 5, type_forecast=c("mean", "median"))
{
 type_forecast <- match.arg(type_forecast)

 if (type_forecast == "mean")
 {
  means <- colMeans(y)
  return(list(mean = t(replicate(n = h, expr = means))))
 } else {
  medians <- apply(y, 2, median)
  return(list(mean = t(replicate(n = h, expr = medians))))
 }

}
```

```
print(fcast_func(x))

res <- crossvalidation::eval_ts(y = x, fcast_func = fcast_func,
fit_params = list(type_forecast = "median"))
colMeans(res)

res <- crossvalidation::eval_ts(y = x, fcast_func = fcast_func,
fit_params = list(type_forecast = "mean"))
colMeans(res)

# Example 8 -----

eval_metric <- function(predicted, observed)
{
  error <- observed - predicted

  res <- apply(error, 2, function(x) sqrt(mean(x ^ 2, na.rm = FALSE)))

  return(res)
}

res <- crossvalidation::eval_ts(y = x, fcast_func = fcast_func,
fit_params = list(type_forecast = "mean"), eval_metric = eval_metric)

colMeans(res)
```

fit_lm *Fit linear model*

Description

Fit linear model

Usage

```
fit_lm(x, y, ...)
```

Arguments

x	design matrix of dimension n * p.
y	vector of observations of length n, or a matrix with n rows.
...	additional parameters to be passed to .lm.fit

Value

a list

Examples

```
NULL
```

`predict_lm` *Linear model prediction*

Description

Linear model prediction

Usage

```
predict_lm(fit_obj, newx)
```

Arguments

`fit_obj`	object adjusted by `crossvalidation::fit_lm`
`newx`	unseen data

Value

a vector or a matrix

Examples

```
NULL
```

`split_ts` *Split a time series*

Description

Split a time series

Usage

```
split_ts(y, p = 0.8, return_indices = FALSE)
```

Arguments

`y`	univariate or multivariate time series
`p`	proportion of data in training set
`return_indices`	return indices instead of time series?

Index