

Package ‘origami’

May 1, 2019

Title Generalized Framework for Cross-Validation

Version 1.0.1

Maintainer Jeremy Coyle <jeremycogle@gmail.com>

Description A general framework for the application of cross-validation schemes to particular functions. By allowing arbitrary lists of results, origami accommodates a range of cross-validation applications.

Depends R (>= 3.0.0),

License GPL-3

URL <https://tlverse.org/origami>

BugReports <https://github.com/tlverse/origami/issues>

Encoding UTF-8

Imports abind, methods, data.table, future, future.apply, listenv

Suggests testthat, class, rmarkdown, knitr, stringr, forecast, randomForest

LazyData true

VignetteBuilder knitr

RoxygenNote 6.1.1

NeedsCompilation no

Author Jeremy Coyle [aut, cre, cph] (<<https://orcid.org/0000-0002-9874-6649>>),
Nima Hejazi [aut] (<<https://orcid.org/0000-0002-7127-2789>>),
Ivana Malenica [ctb] (<<https://orcid.org/0000-0002-7404-8088>>)

Repository CRAN

Date/Publication 2019-05-01 21:50:03 UTC

R topics documented:

Combiners	2
combine_results	3
cross_validate	3

folds2foldvec	6
fold_from_foldvec	6
fold_funs	7
fold_helpers	8
guess_combiner	8
id_folds_to_folds	9
make_fold	9
make_folds	10
make_repeated_folds	10
wrap_in_try	11
Index	12

Combiners

Combiners

Description

Combiners are functions that collapse across a list of similarly structured results. These are standard idioms for combining lists of certain data types.

Usage

```
combiner_rbind(x)
```

```
combiner_c(x)
```

```
combiner_factor(x)
```

```
combiner_array(x)
```

Arguments

x (list) - a list of similar results to be combined.

Value

A combined results object.

combine_results	<i>Combine Results from Different Folds</i>
-----------------	---

Description

Applies [Combiners](#): functions that collapse across a list of similarly structured results, to a list of such lists.

Usage

```
combine_results(results, combiners = NULL, smart_combiners = TRUE)
```

Arguments

results	(list) - a list of lists, corresponding to each result, with the inner lists corresponding to results from each fold.
combiners	(list) - a list with the same names results, containing combiner function names or functions for each result.
smart_combiners	(logical) - if combiners are missing, should they be guessed from the data type of the results.

Details

In theory you should never call this function directly, because it is called automatically by `cross_validate`. The defaults, combiners guessed based on data type, should work in most cases.

Value

A list of combined results.

See Also

[Combiners](#)

cross_validate	<i>Main Cross-Validation Function</i>
----------------	---------------------------------------

Description

Applies `cv_fun` to the folds using `future_lapply` and combines the results across folds using `combine_results`.

Usage

```
cross_validate(cv_fun, folds, ..., use_future = TRUE, .combine = TRUE,  
              .combine_control = list(), .old_results = NULL)
```

Arguments

cv_fun	a function that takes a 'fold' as it's first argument and returns a list of results from that fold. NOTE: the use of an argument named 'X' is specifically disallowed in any input function for compliance with the functions lapply and future.apply::future_lapply.
folds	a list of folds to loop over generated using <code>make_folds</code> .
...	other arguments passed to cvfun.
use_future	logical option for whether to run the main loop of cross-validation with <code>future_lapply</code> or with <code>lapply</code> .
.combine	(logical) - should <code>combine_results</code> be called.
.combine_control	(list) - arguments to <code>combine_results</code> .
.old_results	(list) - the returned result from a previous call to This function. Will be combined with the current results. This is useful for adding additional CV folds to a results object.

Value

A list of results, combined across folds.

Examples

```
#####
# This example explains how to use the cross_validate function naively.
#####
data(mtcars)

# resubstitution MSE
r <- lm(mpg ~ ., data = mtcars)
mean(resid(r)^2)

# function to calculate cross-validated squared error
cv_lm <- function(fold, data, reg_form) {
  # get name and index of outcome variable from regression formula
  out_var <- as.character(unlist(stringr::str_split(reg_form, " ")[1]))
  out_var_ind <- as.numeric(which(colnames(data) == out_var))

  # split up data into training and validation sets
  train_data <- training(data)
  valid_data <- validation(data)

  # fit linear model on training set and predict on validation set
  mod <- lm(as.formula(reg_form), data = train_data)
  preds <- predict(mod, newdata = valid_data)

  # capture results to be returned as output
  out <- list(coef = data.frame(t(coef(mod))),
             SE = ((preds - valid_data[, out_var_ind])^2))
  return(out)
}
```

```

}

# replicate the resubstitution estimate
resub <- make_folds(mtcars, fold_fun = folds_resubstitution)[[1]]
resub_results <- cv_lm(fold = resub, data = mtcars, reg_form = "mpg ~ .")
mean(resub_results$SE)

# cross-validated estimate
folds <- make_folds(mtcars)
cv_results <- cross_validate(cv_fun = cv_lm, folds = folds, data = mtcars,
                             reg_form = "mpg ~ .")
mean(cv_results$SE)
#####
# This example explains how to use the cross_validate function with
# parallelization using the framework of the future package.
#####

suppressMessages(library(data.table))
library(future)
data(mtcars)
set.seed(1)

# make a lot of folds
folds <- make_folds(mtcars, fold_fun = folds_bootstrap, V = 1000)

# function to calculate cross-validated squared error for linear regression
cv_lm <- function(fold, data, reg_form) {
  # get name and index of outcome variable from regression formula
  out_var <- as.character(unlist(str_split(reg_form, " ")[1]))
  out_var_ind <- as.numeric(which(colnames(data) == out_var))

  # split up data into training and validation sets
  train_data <- training(data)
  valid_data <- validation(data)

  # fit linear model on training set and predict on validation set
  mod <- lm(as.formula(reg_form), data = train_data)
  preds <- predict(mod, newdata = valid_data)

  # capture results to be returned as output
  out <- list(coef = data.frame(t(coef(mod))),
              SE = ((preds - valid_data[, out_var_ind])^2))
  return(out)
}

plan(sequential)
time_seq <- system.time({
  results_seq <- cross_validate(cv_fun = cv_lm, folds = folds, data = mtcars,
                               reg_form = "mpg ~ .")
})

plan(multicore)
time_mc <- system.time({

```

```

results_mc <- cross_validate(cv_fun = cv_lm, folds = folds, data = mtcars,
                             reg_form = "mpg ~ .")
})

if(availableCores() > 1) {
  time_mc["elapsed"] < 1.2 * time_seq["elapsed"]
}

```

folds2foldvec

Build a Fold Vector from a Fold Object

Description

For V-fold type cross-validation. This takes a fold object and returns a fold vector (validation set IDs) for use with other tools like `cv.glmnet`.

Usage

```
fold
```

s2foldvec(folds)

Arguments

`folds` A fold object as produced by `make_folds`, from which a numeric vector of the validation set fold IDs are returned.

See Also

Other fold generation functions: [fold_from_foldvec](#), [fold_funs](#), [make_folds](#), [make_repeated_folds](#)

fold_from_foldvec

Build a Fold Object from a Fold Vector

Description

For V-fold type cross-validation. This takes a fold vector (validation set IDs) and builds a fold object for fold V.

Usage

```
fold_from_foldvec(v, folds)
```

Arguments

`v` An identifier of the fold in which observations fall for CV.
`folds` A vector of the fold status for each observation for CV.

See Also

Other fold generation functions: [fold_funs](#), [folds2foldvec](#), [make_folds](#), [make_repeated_folds](#)

Description

These functions represent different cross-validation schemes that can be used with origami. They should be used as options for the `fold_fun` argument to the `make_folds` function in this package. `make_folds` will call the requested function specify `n`, based on its arguments, and pass any remaining arguments (e.g. `V` or `pvalidation`) on.

Usage

```
folds_vfold(n, V = 10)
```

```
folds_resubstitution(n)
```

```
folds_loo(n)
```

```
folds_montecarlo(n, V = 1000, pvalidation = 0.2)
```

```
folds_bootstrap(n, V = 1000)
```

```
folds_rolling_origin(n, first_window, validation_size, gap = 0,  
  batch = 1)
```

```
folds_rolling_window(n, window_size, validation_size, gap = 0,  
  batch = 1)
```

Arguments

`n` (integer) - number of observations.

`V` (integer) - number of folds.

`pvalidation` (double) - proportion of observation to be in validation fold.

`first_window` (integer) - number of observations in the first training sample.

`validation_size` (integer) - number of points in the validation samples; should be equal to the largest forecast horizon.

`gap` (integer) - number of points not included in the training or validation samples; Default is 0.

`batch` (integer) - Increases the number of time-points added to the training set each CV iteration. Applicable for larger time-series. Default is 1.

`window_size` (integer) - number of observations in each training sample.

Value

A list of Folds.

See Also

Other fold generation functions: [fold_from_foldvec](#), [folds2foldvec](#), [make_folds](#), [make_repeated_folds](#)

fold_helpers

Fold Helpers

Description

Accessors and indexers for the different parts of a fold.

Usage

```
training(x = NULL, fold = NULL)
```

```
validation(x = NULL, fold = NULL)
```

```
fold_index(x = NULL, fold = NULL)
```

Arguments

x	an object to be indexed by a training set, validation set, or fold index. If missing, the index itself will be returned.
fold	Fold; the fold used to do the indexing. If missing, fold will be pulled from the calling environment, if available.

Value

The elements of x corresponding to the indexes, or the indexes themselves if x is missing.

See Also

[make_fold](#)

guess_combiner

Flexible Guessing and Mapping for Combining Data Types

Description

Maps data types into standard combiners that should be sensible.

Usage

```
guess_combiner(result)
```


Arguments

result - a single result; flexibly accepts several object classes.

Value

A function to combine a list of such results.

id_folds_to_folds *Convert ID Folds to Observation Folds*

Description

This function convertsf olds that subset ids to folds that subset observations

Usage

`id_folds_to_folds(idfolds, cluster_ids)`

Arguments

`idfolds` folds that subset ids
`cluster_ids` a vector of cluster ids indicating which observations are in which clusters

make_fold *Fold*

Description

Functions to make a fold. Current representation is a simple list.

Usage

`make_fold(v, training_set, validation_set)`

Arguments

`v` integer index of fold in larger scheme.
`training_set` - integer vector of indexes corresponding to the training set.
`validation_set` - integer vector of indexes corresponding to the validation set.

Value

A list containing these elements.

See Also

[fold_helpers](#)

make_folds *Make List of Folds for cross-validation*

Description

Generates a list of folds for a variety of cross-validation schemes.

Usage

```
make_folds(n = NULL, fold_fun = folds_vfold, cluster_ids = NULL,  
           strata_ids = NULL, ...)
```

Arguments

n - either an integer indicating the number of observations to cross-validate over, or an object from which to guess the number of observations; can also be computed from `strata_ids` or `cluster_ids`.

fold_fun - A function indicating the cross-validation scheme to use. See [fold_funs](#) for a list of possibilities.

cluster_ids - a vector of cluster ids. Clusters are treated as a unit - that is, all observations within a cluster are placed in either the training or validation set.

strata_ids - a vector of strata ids. Strata are balanced: insofar as possible the distribution in the sample should be the same as the distribution in the training and validation sets.

... other arguments to be passed to `fold_fun`.

Value

A list of folds objects. Each fold consists of a list with a training index vector, a validation index vector, and a `fold_index` (its order in the list of folds).

See Also

Other fold generation functions: [fold_from_foldvec](#), [fold_funs](#), [folds2foldvec](#), [make_repeated_folds](#)

make_repeated_folds *Repeated Cross-Validation*

Description

Implementation of repeated window cross-validation: generates fold objects for repeated cross-validation by making repeated calls to [make_folds](#) and concatenating the results.

Usage

```
make_repeated_folds(repeats, ...)
```

Arguments

repeats	integer; number of repeats
...	arguments passed to make_folds

See Also

Other fold generation functions: [fold_from_foldvec](#), [fold_funs](#), [folds2foldvec](#), [make_folds](#)

wrap_in_try

Wrap a Function in a Try Statement

Description

Function factory that generates versions of functions wrapped in try.

Usage

```
wrap_in_try(fun, ...)
```

Arguments

fun	A function to be wrapped in a try statement.
...	Additional arguments passed to the previous argument fun.

Index

`combine_results`, [3](#), [4](#)
`combiner_array` (`Combiners`), [2](#)
`combiner_c` (`Combiners`), [2](#)
`combiner_factor` (`Combiners`), [2](#)
`combiner_rbind` (`Combiners`), [2](#)
`Combiners`, [2](#), [3](#)
`cross_validate`, [3](#)

`fold_from_foldvec`, [6](#), [6](#), [8](#), [10](#), [11](#)
`fold_funs`, [6](#), [7](#), [10](#), [11](#)
`fold_helpers`, [8](#), [9](#)
`fold_index` (`fold_helpers`), [8](#)
`folds2foldvec`, [6](#), [6](#), [8](#), [10](#), [11](#)
`folds_bootstrap` (`fold_funs`), [7](#)
`folds_loo` (`fold_funs`), [7](#)
`folds_montecarlo` (`fold_funs`), [7](#)
`folds_resubstitution` (`fold_funs`), [7](#)
`folds_rolling_origin` (`fold_funs`), [7](#)
`folds_rolling_window` (`fold_funs`), [7](#)
`folds_vfold` (`fold_funs`), [7](#)

`guess_combiner`, [8](#)

`id_folds_to_folds`, [9](#)

`make_fold`, [8](#), [9](#)
`make_folds`, [4](#), [6–8](#), [10](#), [10](#), [11](#)
`make_repeated_folds`, [6](#), [8](#), [10](#), [10](#)

`training` (`fold_helpers`), [8](#)

`validation` (`fold_helpers`), [8](#)

`wrap_in_try`, [11](#)