Package: mlr3resampling (via r-universe)

November 5, 2024

Type Package

Title Resampling Algorithms for 'mlr3' Framework

Version 2024.10.28

Description A supervised learning algorithm inputs a train set, and outputs a prediction function, which can be used on a test set. If each data point belongs to a subset (such as geographic region, year, etc), then how do we know if subsets are similar enough so that we can get accurate predictions on one subset, after training on Other subsets? And how do we know if training on All subsets would improve prediction accuracy, relative to training on the Same subset? SOAK, Same/Other/All K-fold cross-validation, <doi:10.48550/arXiv.2410.08643> can be used to answer these question, by fixing a test subset, training models on Same/Other/All subsets, and then comparing test error rates (Same versus Other and Same versus All). Also provides code for estimating how many train samples are required to get accurate predictions on a test set.

License GPL-3

URL https://github.com/tdhock/mlr3resampling

BugReports https://github.com/tdhock/mlr3resampling/issues

Imports data.table, R6, checkmate, paradox, mlr3 (>= 0.21.1), mlr3misc

Suggests ggplot2, animint2, mlr3tuning, lgr, future, testthat, knitr, markdown, nc, rpart, directlabels

VignetteBuilder knitr

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

RemoteUrl https://github.com/tdhock/mlr3resampling

RemoteRef HEAD

RemoteSha e6e3406032ed4693afab549398ba46da0ff5abda

Contents

AZtrees . . . 2
ResamplingSameOtherCV . . . 3
ResamplingSameOtherSizesCV . . . 6
ResamplingVariableSizeTrainCV . . . 8
score . . . 10

Index **13**

`AZtrees` *Arizona Trees*

Description

Classification data set with polygons (groups which should not be split in CV) and subsets (region3 or region4).

Usage

```
data("AZtrees")
```

Format

A data frame with 5956 observations on the following 25 variables.

`region3` a character vector

`region4` a character vector

`polygon` a numeric vector

`y` a character vector

`ycoord` latitude

`xcoord` longitude

`SAMPLE_1` a numeric vector

`SAMPLE_2` a numeric vector

`SAMPLE_3` a numeric vector

`SAMPLE_4` a numeric vector

`SAMPLE_5` a numeric vector

`SAMPLE_6` a numeric vector

`SAMPLE_7` a numeric vector

`SAMPLE_8` a numeric vector

`SAMPLE_9` a numeric vector

`SAMPLE_10` a numeric vector

`SAMPLE_11` a numeric vector

SAMPLE_12 a numeric vector

SAMPLE_13 a numeric vector

SAMPLE_14 a numeric vector

SAMPLE_15 a numeric vector

SAMPLE_16 a numeric vector

SAMPLE_17 a numeric vector

SAMPLE_18 a numeric vector

SAMPLE_19 a numeric vector

SAMPLE_20 a numeric vector

SAMPLE_21 a numeric vector

Source

Paul Nelson Arellano, paul.arellano@nau.edu

Examples

```
data(AZtrees)
task.obj <- mlr3::TaskClassif$new("AZtrees3", AZtrees, target="y")
task.obj$col_roles$feature <- grep("SAMPLE", names(AZtrees), value=TRUE)
task.obj$col_roles$group <- "polygon"
task.obj$col_roles$subset <- "region3"
str(task.obj)
same_other_sizes_cv <- mlr3resampling::ResamplingSameOtherSizesCV$new()
same_other_sizes_cv$instantiate(task.obj)
same_other_sizes_cv$instance$iteration.dt
```

ResamplingSameOtherCV *Resampling for comparing training on same or other subsets*

Description

ResamplingSameOtherCV defines how a task is partitioned for resampling, for example in resample() or benchmark().

Resampling objects can be instantiated on a Task, which should define at least one subset variable.

After instantiation, sets can be accessed via $train_set(i) and $test_set(i), respectively.

Details

This provides an implementation of SOAK, Same/Other/All K-fold cross-validation. After instantiation, this class provides information in `$instance` that can be used for visualizing the splits, as shown in the vignette. Most typical machine learning users should instead use `ResamplingSameOtherSizesCV`, which does not support these visualization features, but provides other relevant machine learning features, such as group role, which is not supported by `ResamplingSameOtherCV`.

A supervised learning algorithm inputs a train set, and outputs a prediction function, which can be used on a test set. If each data point belongs to a subset (such as geographic region, year, etc), then how do we know if it is possible to train on one subset, and predict accurately on another subset? Cross-validation can be used to determine the extent to which this is possible, by first assigning fold IDs from 1 to K to all data (possibly using stratification, usually by subset and label). Then we loop over test sets (subset/fold combinations), train sets (same subset, other subsets, all subsets), and compute test/prediction accuracy for each combination. Comparing test/prediction accuracy between same and other, we can determine the extent to which it is possible (perfect if same/other have similar test accuracy for each subset; other is usually somewhat less accurate than same; other can be just as bad as featureless baseline when the subsets have different patterns).

Stratification

`ResamplingSameOtherCV` supports stratified sampling. The stratification variables are assumed to be discrete, and must be stored in the Task with column role `"stratum"`. In case of multiple stratification variables, each combination of the values of the stratification variables forms a stratum.

Grouping

`ResamplingSameOtherCV` does not support grouping of observations that should not be split in cross-validation. See `ResamplingSameOtherSizesCV` for another sampler which does support both `group` and `subset` roles.

Subsets

The subset variable is assumed to be discrete, and must be stored in the Task with column role `"subset"`. The number of cross-validation folds K should be defined as the `fold` parameter. In each subset, there will be about an equal number of observations assigned to each of the K folds. The assignments are stored in `$instance$id.dt`. The train/test splits are defined by all possible combinations of test subset, test fold, and train subsets (Same/Other/All). The splits are stored in `$instance$iteration.dt`.

Methods

Public methods:

- `Resampling$new()`
- `Resampling$train_set()`
- `Resampling$test_set()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
Resampling$new(
  id,
  param_set = ps(),
  duplicated_ids = FALSE,
  label = NA_character_,
  man = NA_character_
)
```

Arguments:

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

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

`duplicated_ids` (`logical(1)`)
: Set to `TRUE` if this resampling strategy may have duplicated row ids in a single training set or test set.

`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()`.

Method `train_set()`**:** Returns the row ids of the i-th training set.

Usage:

```
Resampling$train_set(i)
```

Arguments:

`i` (`integer(1)`)
: Iteration.

Returns: (`integer()`) of row ids.

Method `test_set()`**:** Returns the row ids of the i-th test set.

Usage:

```
Resampling$test_set(i)
```

Arguments:

`i` (`integer(1)`)
: Iteration.

Returns: (`integer()`) of row ids.

See Also

- arXiv paper `https://arxiv.org/abs/2410.08643` describing SOAK algorithm.
- Articles `https://github.com/tdhock/mlr3resampling/wiki/Articles`
- Package **mlr3** for standard `Resampling`, which does not support comparing train on Same/Other/All subsets.
- `vignette(package="mlr3resampling")` for more detailed examples.

Examples

```
same_other <- mlr3resampling::ResamplingSameOtherCV$new()
same_other$param_set$values$folds <- 5
```

ResamplingSameOtherSizesCV

Resampling for comparing train subsets and sizes

Description

ResamplingSameOtherSizesCV defines how a task is partitioned for resampling, for example in resample() or benchmark().

Resampling objects can be instantiated on a Task, which can use the subset role.

After instantiation, sets can be accessed via $train_set(i) and $test_set(i), respectively.

Details

This is an implementation of SOAK, Same/Other/All K-fold cross-validation. A supervised learning algorithm inputs a train set, and outputs a prediction function, which can be used on a test set. If each data point belongs to a subset (such as geographic region, year, etc), then how do we know if it is possible to train on one subset, and predict accurately on another subset? Cross-validation can be used to determine the extent to which this is possible, by first assigning fold IDs from 1 to K to all data (possibly using stratification, usually by subset and label). Then we loop over test sets (subset/fold combinations), train sets (same subset, other subsets, all subsets), and compute test/prediction accuracy for each combination. Comparing test/prediction accuracy between same and other, we can determine the extent to which it is possible (perfect if same/other have similar test accuracy for each subset; other is usually somewhat less accurate than same; other can be just as bad as featureless baseline when the subsets have different patterns).

This class has more parameters/potential applications than ResamplingSameOtherCV and ResamplingVariableSizeTrainC which are older and should only be preferred for visualization purposes.

Stratification

ResamplingSameOtherSizesCV supports stratified sampling. The stratification variables are assumed to be discrete, and must be stored in the Task with column role "stratum". In case of multiple stratification variables, each combination of the values of the stratification variables forms a stratum.

Grouping

ResamplingSameOtherSizesCV supports grouping of observations that will not be split in cross-validation. The grouping variable is assumed to be discrete, and must be stored in the Task with column role "group".

Subsets

ResamplingSameOtherSizesCV supports training on different subsets of observations. The subset variable is assumed to be discrete, and must be stored in the Task with column role "subset".

Parameters

The number of cross-validation folds K should be defined as the `fold` parameter, default 3.

The number of random seeds for down-sampling should be defined as the `seeds` parameter, default 1.

The ratio for down-sampling should be defined as the `ratio` parameter, default 0.5. The min size of same and other sets is repeatedly multiplied by this ratio, to obtain smaller sample sizes.

The number of down-sampling sizes/multiplications should be defined as the `sizes` parameter, which can also take two special values: default -1 means no down-sampling at all, and 0 means only down-sampling to the sizes of the same/other sets.

The `ignore_subset` parameter should be either `TRUE` or `FALSE` (default), whether to ignore the `subset` role. `TRUE` only creates splits for same subset (even if task defines `subset` role), and is useful for subtrain/validation splits (hyper-parameter learning). Note that this feature will work on a task with both `stratum` and `group` roles (unlike `ResamplingCV`).

In each subset, there will be about an equal number of observations assigned to each of the K folds. The train/test splits are defined by all possible combinations of test subset, test fold, train subsets (same/other/all), down-sampling sizes, and random seeds. The splits are stored in `$instance$iteration.dt`.

Methods

Public methods:

- `Resampling$new()`
- `Resampling$train_set()`
- `Resampling$test_set()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
Resampling$new(
  id,
  param_set = ps(),
  duplicated_ids = FALSE,
  label = NA_character_,
  man = NA_character_
)
```

Arguments:

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

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

`duplicated_ids` (`logical(1)`)
Set to `TRUE` if this resampling strategy may have duplicated row ids in a single training set or test set.

`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()`.

Method `train_set()`**:** Returns the row ids of the i-th training set.

Usage:

```
Resampling$train_set(i)
```

Arguments:

`i` (`integer(1)`)
Iteration.

Returns: (`integer()`) of row ids.

Method `test_set()`**:** Returns the row ids of the i-th test set.

Usage:

```
Resampling$test_set(i)
```

Arguments:

`i` (`integer(1)`)
Iteration.

Returns: (`integer()`) of row ids.

See Also

- arXiv paper https://arxiv.org/abs/2410.08643 describing SOAK algorithm.
- Articles https://github.com/tdhock/mlr3resampling/wiki/Articles
- Package **mlr3** for standard `Resampling`, which does not support comparing train on Same/Other/All subsets.
- `vignette(package="mlr3resampling")` for more detailed examples.

Examples

```
same_other_sizes <- mlr3resampling::ResamplingSameOtherSizesCV$new()
same_other_sizes$param_set$values$folds <- 5
```

ResamplingVariableSizeTrainCV
Resampling for comparing training on same or other groups

Description

`ResamplingVariableSizeTrainCV` defines how a task is partitioned for resampling, for example in `resample()` or `benchmark()`.

Resampling objects can be instantiated on a `Task`.

After instantiation, sets can be accessed via `$train_set(i)` and `$test_set(i)`, respectively.

Details

A supervised learning algorithm inputs a train set, and outputs a prediction function, which can be used on a test set. How many train samples are required to get accurate predictions on a test set? Cross-validation can be used to answer this question, with variable size train sets.

Stratification

`ResamplingVariableSizeTrainCV` supports stratified sampling. The stratification variables are assumed to be discrete, and must be stored in the Task with column role `"stratum"`. In case of multiple stratification variables, each combination of the values of the stratification variables forms a stratum.

Grouping

`ResamplingVariableSizeTrainCV` does not support grouping of observations.

Hyper-parameters

The number of cross-validation folds should be defined as the `fold` parameter.

For each fold ID, the corresponding observations are considered the test set, and a variable number of other observations are considered the train set.

The `random_seeds` parameter controls the number of random orderings of the train set that are considered.

For each random order of the train set, the `min_train_data` parameter controls the size of the smallest stratum in the smallest train set considered.

To determine the other train set sizes, we use an equally spaced grid on the log scale, from `min_train_data` to the largest train set size (all data not in test set). The number of train set sizes in this grid is determined by the `train_sizes` parameter.

Methods

Public methods:

- `Resampling$new()`
- `Resampling$train_set()`
- `Resampling$test_set()`

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
Resampling$new(
  id,
  param_set = ps(),
  duplicated_ids = FALSE,
  label = NA_character_,
  man = NA_character_
)
```

Arguments:

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

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

`duplicated_ids` (`logical(1)`)
: Set to `TRUE` if this resampling strategy may have duplicated row ids in a single training set or test set.

`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()`.

Method `train_set()`**:** Returns the row ids of the i-th training set.

Usage:

```
Resampling$train_set(i)
```

Arguments:

`i` (`integer(1)`)
: Iteration.

Returns: (`integer()`) of row ids.

Method `test_set()`**:** Returns the row ids of the i-th test set.

Usage:

```
Resampling$test_set(i)
```

Arguments:

`i` (`integer(1)`)
: Iteration.

Returns: (`integer()`) of row ids.

Examples

```
(var_sizes <- mlr3resampling::ResamplingVariableSizeTrainCV$new())
```

`score` *Score benchmark results*

Description

Computes a data table of scores.

Usage

```
score(bench.result, ...)
```

Arguments

bench.result	Output of benchmark().
...	Additional arguments to pass to bench.result$score, for example measures.

Value

data table with scores.

Author(s)

Toby Dylan Hocking

Examples

```
N <- 100
library(data.table)
set.seed(1)
reg.dt <- data.table(
  x=runif(N, -2, 2),
  person=rep(1:2, each=0.5*N))
reg.pattern.list <- list(
  easy=function(x, person)x^2,
  impossible=function(x, person)(x^2+person*3)*(-1)^person)
reg.task.list <- list()
for(pattern in names(reg.pattern.list)){
  f <- reg.pattern.list[[pattern]]
  yname <- paste0("y_",pattern)
  reg.dt[, (yname) := f(x,person)+rnorm(N, sd=0.5)][]
  task.dt <- reg.dt[, c("x","person",yname), with=FALSE]
  task.obj <- mlr3::TaskRegr$new(
    pattern, task.dt, target=yname)
  task.obj$col_roles$stratum <- "person"
  task.obj$col_roles$subset <- "person"
  reg.task.list[[pattern]] <- task.obj
}
same_other <- mlr3resampling::ResamplingSameOtherSizesCV$new()
reg.learner.list <- list(
  mlr3::LearnerRegrFeatureless$new())
if(requireNamespace("rpart")){
  reg.learner.list$rpart <- mlr3::LearnerRegrRpart$new()
}
(bench.grid <- mlr3::benchmark_grid(
  reg.task.list,
  reg.learner.list,
  same_other))
bench.result <- mlr3::benchmark(bench.grid)
bench.score <- mlr3resampling::score(bench.result)
if(require(animint2)){
  ggplot()+
    geom_point(aes(
      regr.mse, train.subsets, color=algorithm),
```

```
      shape=1,
      data=bench.score)+
    facet_grid(
      test.subset ~ task_id,
      labeller=label_both,
      scales="free")+
    scale_x_log10()
}
```

Index

∗ **Resampling**
    ResamplingSameOtherCV, 3
    ResamplingSameOtherSizesCV, 6
    ResamplingVariableSizeTrainCV, 8
∗ **datasets**
    AZtrees, 2

AZtrees, 2

benchmark(), *3*, *6*, *8*, *11*

paradox::ParamSet, *5*, *7*, *10*

R6, *4*, *7*, *9*
resample(), *3*, *6*, *8*
Resampling, *5*, *8*
ResamplingSameOtherCV, *3*, 3, *4*, *6*
ResamplingSameOtherSizesCV, *4*, *6*, 6, *7*
ResamplingVariableSizeTrainCV, *6*, *8*, 8, *9*

score, 10

Task, *3*, *4*, *6–9*