Package ‘mlr3resampling’

June 23, 2025

Type Package

Title Resampling Algorithms for 'mlr3' Framework

Version 2025.6.23

Encoding UTF-8

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 questions, 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 LGPL-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, batchtools, filelock

Suggests ggplot2, animint2, mlr3tuning, lgr, future, testthat, knitr,
markdown, nc, rpart, directlabels, mlr3torch, torch

VignetteBuilder knitr

NeedsCompilation no

Author Toby Hocking [aut, cre] (ORCID:
<`https://orcid.org/0000-0002-3146-0865`>),
Michel Lang [ctb] (ORCID: <`https://orcid.org/0000-0001-9754-0393`>,
Author of mlr3 when Resampling/ResamplingCV was copied/modified),

Bernd Bischl [ctb] (ORCID: <https://orcid.org/0000-0001-6002-6980>,
Author of mlr3 when Resampling/ResamplingCV was copied/modified),
Jakob Richter [ctb] (ORCID: <https://orcid.org/0000-0003-4481-5554>,
Author of mlr3 when Resampling/ResamplingCV was copied/modified),
Patrick Schratz [ctb] (ORCID: <https://orcid.org/0000-0003-0748-6624>,
Author of mlr3 when Resampling/ResamplingCV was copied/modified),
Giuseppe Casalicchio [ctb] (ORCID:
<https://orcid.org/0000-0001-5324-5966>, Author of mlr3 when
Resampling/ResamplingCV was copied/modified),
Stefan Coors [ctb] (ORCID: <https://orcid.org/0000-0002-7465-2146>,
Author of mlr3 when Resampling/ResamplingCV was copied/modified),
Quay Au [ctb] (ORCID: <https://orcid.org/0000-0002-5252-8902>, Author
of mlr3 when Resampling/ResamplingCV was copied/modified),
Martin Binder [ctb],
Florian Pfisterer [ctb] (ORCID:
<https://orcid.org/0000-0001-8867-762X>, Author of mlr3 when
Resampling/ResamplingCV was copied/modified),
Raphael Sonabend [ctb] (ORCID: <https://orcid.org/0000-0001-9225-4654>,
Author of mlr3 when Resampling/ResamplingCV was copied/modified),
Lennart Schneider [ctb] (ORCID:
<https://orcid.org/0000-0003-4152-5308>, Author of mlr3 when
Resampling/ResamplingCV was copied/modified),
Marc Becker [ctb] (ORCID: <https://orcid.org/0000-0002-8115-0400>,
Author of mlr3 when Resampling/ResamplingCV was copied/modified),
Sebastian Fischer [ctb] (ORCID:
<https://orcid.org/0000-0002-9609-3197>, Author of mlr3 when
Resampling/ResamplingCV was copied/modified)

Maintainer Toby Hocking <toby.hocking@r-project.org>

Repository CRAN

Date/Publication 2025-06-23 10:20:02 UTC

Contents

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
```

---

proj_compute *Compute resampling results in a project*

Description

`proj_compute()` looks in `grid_jobs.csv` for a row with `status=="not started"`, and picks the first one to work on, updating `status="started"`. After having run `train()` and `predict()`, a data table with one row is saved to an RDS file in the `grid_jobs` directory, and `status="done"` is updated. `proj_compute_until_done()` keeps doing that in a while loop.

Usage

```
proj_compute(proj_dir, verbose = FALSE)
proj_compute_until_done(proj_dir, verbose = FALSE)
```

Arguments

`proj_dir`	Project directory created by `proj_grid`.
`verbose`	Logical: print messages?

Details

If everything goes well, the user should not need to run this function. Instead, the user runs `proj_submit` as Step 2 out of the typical 3 step pipeline (init grid, submit, read results). `proj_compute` can sometimes be useful for testing or debugging the submit step, since it runs one split at a time.

Value

Data table with one row of results.

Author(s)

Toby Dylan Hocking

Examples

```
N <- 80
library(data.table)
set.seed(1)
reg.dt <- data.table(
  x=runif(N, -2, 2),
  person=factor(rep(c("Alice","Bob"), each=0.5*N)))
reg.pattern.list <- list(
  easy=function(x, person)x^2,
  impossible=function(x, person)(x^2)*(-1)^as.integer(person))
SOAK <- mlr3resampling::ResamplingSameOtherSizesCV$new()
reg.task.list <- list()
for(pattern in names(reg.pattern.list)){
  f <- reg.pattern.list[[pattern]]
  task.dt <- data.table(reg.dt)[
  , y := f(x,person)+rnorm(N, sd=0.5)
  ][]
  task.obj <- mlr3::TaskRegr$new(
    pattern, task.dt, target="y")
  task.obj$col_roles$feature <- "x"
  task.obj$col_roles$stratum <- "person"
  task.obj$col_roles$subset <- "person"
  reg.task.list[[pattern]] <- task.obj
}
reg.learner.list <- list(
  featureless=mlr3::LearnerRegrFeatureless$new())
if(requireNamespace("rpart")){
  reg.learner.list$rpart <- mlr3::LearnerRegrRpart$new()
}

pkg.proj.dir <- tempfile()
mlr3resampling::proj_grid(
  pkg.proj.dir,
  reg.task.list,
  reg.learner.list,
  SOAK,
  order_jobs = function(DT)1:2, # for CRAN.
  score_args=mlr3::msrs(c("regr.rmse", "regr.mae")))
mlr3resampling::proj_compute(pkg.proj.dir)
fread(file.path(pkg.proj.dir, "grid_jobs.csv"))
mlr3resampling::proj_compute_until_done(pkg.proj.dir)
fread(file.path(pkg.proj.dir, "grid_jobs.csv"))
```

`proj_grid` *Initialize a new project grid table*

Description

A project grid consists of all combinations of tasks, learners, resampling types, and resampling iterations, to be computed in parallel. This function creates a project directory with files to describe the grid.

Usage

```
proj_grid(
  proj_dir, tasks, learners, resamplings,
  order_jobs = NULL, score_args = NULL,
  save_learner = FALSE, save_pred = FALSE)
```

Arguments

`proj_dir`	Path to directory to create.
`tasks`	List of Tasks, or a single Task.
`learners`	List of Learners, or a single Learner.
`resamplings`	List of Resamplings, or a single Resampling.
`order_jobs`	Function which takes split table as input, and returns integer vector of row numbers of the split table to write to `grid_jobs.csv`, which is how worker processes determine what work to do next (smaller numbers have higher priority). Default `NULL` means to keep default order.
`score_args`	Passed to `pred$score()`.
`save_learner`	Function to process Learner, after training/prediction, but before saving result to disk. For interpreting complex models, you should write a function that returns only the parts of the model that you need (and discards the other parts which would take up disk space for no reason). Default `FALSE` means to not keep it (always returns `NULL`). `TRUE` means to keep it without any special processing.
`save_pred`	Function to process Prediction before saving to disk. Default `FALSE` means to not keep it (always returns `NULL`). `TRUE` means to keep it without any special processing.

Details

This is Step 1 out of the typical 3 step pipeline (init grid, submit, read results). It creates a `grid_jobs.csv` table which has a column `status`; each row is initialized to `"not started"` or `"done"`, depending on whether the corresponding result RDS file exists already.

Value

Data table of splits to be processed (same as table saved to `grid_jobs.rds`).

Author(s)

Toby Dylan Hocking

Examples

```
N <- 80
library(data.table)
set.seed(1)
reg.dt <- data.table(
  x=runif(N, -2, 2),
  person=factor(rep(c("Alice","Bob"), each=0.5*N)))
reg.pattern.list <- list(
  easy=function(x, person)x^2,
  impossible=function(x, person)(x^2)*(-1)^as.integer(person))
SOAK <- mlr3resampling::ResamplingSameOtherSizesCV$new()
reg.task.list <- list()
for(pattern in names(reg.pattern.list)){
  f <- reg.pattern.list[[pattern]]
  task.dt <- data.table(reg.dt)[
  , y := f(x,person)+rnorm(N, sd=0.5)
  ][]
  task.obj <- mlr3::TaskRegr$new(
    pattern, task.dt, target="y")
  task.obj$col_roles$feature <- "x"
  task.obj$col_roles$stratum <- "person"
  task.obj$col_roles$subset <- "person"
  reg.task.list[[pattern]] <- task.obj
}
reg.learner.list <- list(
  featureless=mlr3::LearnerRegrFeatureless$new())
if(requireNamespace("rpart")){
  reg.learner.list$rpart <- mlr3::LearnerRegrRpart$new()
}

pkg.proj.dir <- tempfile()
mlr3resampling::proj_grid(
  pkg.proj.dir,
  reg.task.list,
  reg.learner.list,
  SOAK,
  score_args=mlr3::msrs(c("regr.rmse", "regr.mae")))
mlr3resampling::proj_compute(pkg.proj.dir)
fread(file.path(pkg.proj.dir, "grid_jobs.csv"))
```

`proj_results` *Combine and save results in a project*

Description

`proj_results` globs the RDS result files in the project directory, and combines them into a result table via `rbindlist()`. `proj_results_save` saves that result table to `results.rds` and `results.csv`.

Usage

```
proj_results(proj_dir)
proj_results_save(proj_dir)
```

Arguments

`proj_dir`	Project directory created via `proj_grid`.

Details

This is Step 3 out of the typical 3 step pipeline (init grid, submit, read results). Actually, if step 2 worked as intended, the last `proj_compute` calls `proj_results_save`, which saves up to three result files to disk that you can read directly:

`results.csv` contains test measures for each split, and can be read via `fread()`

`results.rds` contains additional list columns for `learner` and `pred` (useful for model interpretation), and can be read via `readRDS()`

`learners.csv` only exists if `learner` column is a data frame, in which case it contains the atomic columns, along with meta-data describing each split.

Value

`proj_results` returns a data table with all columns, whereas `proj_results_save` returns the same table with only atomic columns.

Author(s)

Toby Dylan Hocking

Examples

```
N <- 80
library(data.table)
set.seed(1)
reg.dt <- data.table(
  x=runif(N, -2, 2),
  person=factor(rep(c("Alice","Bob"), each=0.5*N)))
reg.pattern.list <- list(
  easy=function(x, person)x^2,
  impossible=function(x, person)(x^2)*(-1)^as.integer(person))
SOAK <- mlr3resampling::ResamplingSameOtherSizesCV$new()
reg.task.list <- list()
for(pattern in names(reg.pattern.list)){
  f <- reg.pattern.list[[pattern]]
  task.dt <- data.table(reg.dt)[
  , y := f(x,person)+rnorm(N, sd=0.5)
  ][]
  task.obj <- mlr3::TaskRegr$new(
    pattern, task.dt, target="y")
  task.obj$col_roles$feature <- "x"
```

```
    task.obj$col_roles$stratum <- "person"
    task.obj$col_roles$subset <- "person"
    reg.task.list[[pattern]] <- task.obj
  }
  reg.learner.list <- list(
    featureless=mlr3::LearnerRegrFeatureless$new())
  if(requireNamespace("rpart")){
    reg.learner.list$rpart <- mlr3::LearnerRegrRpart$new()
  }

  pkg.proj.dir <- tempfile()
  mlr3resampling::proj_grid(
    pkg.proj.dir,
    reg.task.list,
    reg.learner.list,
    SOAK,
    order_jobs = function(DT)1:2, # for CRAN.
    score_args=mlr3::msrs(c("regr.rmse", "regr.mae")))
  mlr3resampling::proj_compute_until_done(pkg.proj.dir)
  fread(file.path(pkg.proj.dir, "results.csv"))
```

`proj_submit` *Submit resampling split jobs in parallel*

Description

Before running this function, you should define `cluster.functions` in your `~/.batchtools.conf.R` file. It makes a batchtools registry, then runs `batchtools::batchMap()` and `batchtools::submitJobs()`; each iteration runs `proj_compute_until_done`.

Usage

```
proj_submit(
  proj_dir, tasks = 2, hours = 1, gigabytes = 1,
  verbose = FALSE, cluster.functions = NULL)
```

Arguments

`proj_dir`	Project directory created via `proj_grid`.
`tasks`	Positive integer: number of batchtools jobs, translated into one SLURM job array with this number of tasks.
`hours`	Hours of walltime to ask the SLURM scheduler.
`gigabytes`	Gigabytes of memory to ask the SLURM scheduler.
`verbose`	Logical: print messages?
`cluster.functions`	Cluster functions from `batchtools`, useful for testing.

Details

This is Step 2 out of the typical 3 step pipeline (init grid, submit, read results).

Value

The batchtools registry.

Author(s)

Toby Dylan Hocking

Examples

```
N <- 80
library(data.table)
set.seed(1)
reg.dt <- data.table(
  x=runif(N, -2, 2),
  person=factor(rep(c("Alice","Bob"), each=0.5*N)))
reg.pattern.list <- list(
  easy=function(x, person)x^2,
  impossible=function(x, person)(x^2)*(-1)^as.integer(person))
SOAK <- mlr3resampling::ResamplingSameOtherSizesCV$new()
reg.task.list <- list()
for(pattern in names(reg.pattern.list)){
  f <- reg.pattern.list[[pattern]]
  task.dt <- data.table(reg.dt)[
  , y := f(x,person)+rnorm(N, sd=0.5)
  ][]
  task.obj <- mlr3::TaskRegr$new(
    pattern, task.dt, target="y")
  task.obj$col_roles$feature <- "x"
  task.obj$col_roles$stratum <- "person"
  task.obj$col_roles$subset <- "person"
  reg.task.list[[pattern]] <- task.obj
}
reg.learner.list <- list(
  featureless=mlr3::LearnerRegrFeatureless$new())
if(requireNamespace("rpart")){
  reg.learner.list$rpart <- mlr3::LearnerRegrRpart$new()
}

pkg.proj.dir <- tempfile()
mlr3resampling::proj_grid(
  pkg.proj.dir,
  reg.task.list,
  reg.learner.list,
  SOAK,
  order_jobs = function(DT)1:2, # for CRAN.
  score_args=mlr3::msrs(c("regr.rmse", "regr.mae")))
mlr3resampling::proj_submit(pkg.proj.dir)
batchtools::waitForJobs()
```

```
fread(file.path(pkg.proj.dir, "results.csv"))
```

---

pvalue *P-values for comparing Same/Other/All training*

Description

Same/Other/All K-fold cross-validation (SOAK) results in K measures of test error/accuracy. This function computes P-values (two-sided T-test) between Same and All/Other.

Usage

```
pvalue(score_in, value.var = NULL, digits=3)
```

Arguments

`score_in`	Data table output from `score`.
`value.var`	Name of column to use as the evaluation metric in T-test. Default NULL means to use the first column matching `"classif|regr"`.
`digits`	Number of decimal places to show for mean and standard deviation.

Value

List of class "pvalue" with named elements `value.var`, `stats`, `pvalues`.

Author(s)

Toby Dylan Hocking

Examples

```
N <- 80
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)*(-1)^person)
SOAK <- mlr3resampling::ResamplingSameOtherSizesCV$new()
reg.task.list <- list()
for(pattern in names(reg.pattern.list)){
  f <- reg.pattern.list[[pattern]]
  yname <- paste0("y_",pattern)
  reg.dt[, (yname) := f(x,person)+rnorm(N, sd=0.5)][]
  task.dt <- reg.dt[, c("x","person",yname), with=FALSE]
  task.obj <- mlr3::TaskRegr$new(
    pattern, task.dt, target=yname)
  task.obj$col_roles$stratum <- "person"
  task.obj$col_roles$subset <- "person"
  reg.task.list[[pattern]] <- task.obj
}
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,
  SOAK))
bench.result <- mlr3::benchmark(bench.grid)
bench.score <- mlr3resampling::score(bench.result, mlr3::msr("regr.rmse"))
bench.plist <- mlr3resampling::pvalue(bench.score)
plot(bench.plist)
```

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`).

The `subsets` parameter should specify the train subsets of interest: `"S"` (same), `"O"` (other), `"A"` (all), `"SO"`, `"SA"`, `"SOA"` (default).

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 <- 80
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)*(-1)^person)
SOAK <- mlr3resampling::ResamplingSameOtherSizesCV$new()
reg.task.list <- list()
for(pattern in names(reg.pattern.list)){
  f <- reg.pattern.list[[pattern]]
  yname <- paste0("y_",pattern)
  reg.dt[, (yname) := f(x,person)+rnorm(N, sd=0.5)][]
  task.dt <- reg.dt[, c("x","person",yname), with=FALSE]
  task.obj <- mlr3::TaskRegr$new(
    pattern, task.dt, target=yname)
  task.obj$col_roles$stratum <- "person"
  task.obj$col_roles$subset <- "person"
  reg.task.list[[pattern]] <- task.obj
}
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,
  SOAK))
bench.result <- mlr3::benchmark(bench.grid)
bench.score <- mlr3resampling::score(bench.result, mlr3::msr("regr.rmse"))
plot(bench.score)
```

Index