Package 'mlr3resampling'

May 19, 2026

Type Package

Title Resampling Algorithms for 'mlr3' Framework

Version 2026.5.19

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:10.1002/sam.70055>
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

LinkingTo Rcpp, RcppArmadillo

Imports data.table, R6, checkmate, paradox, mlr3 (>= 1.0.0), mlr3misc,
methods, Rcpp

Suggests pbdMPI, geepack, ggplot2, mlr3tuning, lgr, future,
future.apply, testthat, WeightedROC, nc, rpart, directlabels,
mlr3pipelines, glmnet, mlr3learners, mlr3torch, torch,
batchtools, mlr3batchmark, litedown

VignetteBuilder litedown

NeedsCompilation yes

Author Toby Hocking [aut, cre] (ORCID:
<https://orcid.org/0000-0002-3146-0865>),
Daniel Agyapong [ctb] (ORCID: <https://orcid.org/0009-0004-0857-3150>),
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 2026-05-19 15:20:02 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.table::setDTthreads(1L)
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
```

Learners *Learner classes with special methods*

Description

AutoTunerTorch_epochs inherits from mlr3tuning::AutoTuner, with an initialize method that takes arguments to construct a torch module learner. It runs gradient descent up to max_epochs and then re-runs using the best number of epochs. Its edit_learner method sets number of epochs to 2 (for quick learning during proj_test), and its save_learner method returns a history data table (one row per epoch). LearnerRegrCVGlmnetSave inherits from LearnerRegrCVGlmnet; its save_learner method returns a data table of regularized linear model weights (no edit_learner method). LearnerClassifCVGlmnetSave is similar.

Author(s)

Toby Dylan Hocking

Examples

```
## Simulate regression data.
N <- 80
library(data.table)
set.seed(1)
reg.dt <- data.table(
  x=runif(N, -2, 2),
  noise=runif(N, -2, 2),
  person=factor(rep(c("Alice","Bob"), each=0.5*N)))
reg.pattern.list <- list(
  easy=function(x, person)x^3,
  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 <- c("x","noise")
  task.obj$col_roles$stratum <- "person"
  task.obj$col_roles$subset <- "person"
  reg.task.list[[pattern]] <- task.obj
}
reg.task.list # two regression tasks.

## Create a list of learners.
reg.learner.list <- list(
  featureless=mlr3::LearnerRegrFeatureless$new())
if(requireNamespace("mlr3torch") && torch::torch_is_installed()){
  gen_linear <- torch::nn_module(
    "my_linear",
    initialize = function(task) {
      self$weight = torch::nn_linear(task$n_features, 1)
    },
    forward = function(x) {
      self$weight(x)
    }
  )
  reg.learner.list$torch_linear <- mlr3resampling::AutoTunerTorch_epochs$new(
    "torch_linear",
    module_generator=gen_linear,
    max_epochs=3,
    batch_size=10,
    loss=mlr3torch::t_loss("mse"),
    measure_list=mlr3::msrs(c("regr.mse","regr.mae")))
}
if(requireNamespace("mlr3learners")){
  reg.learner.list$cv_glmnet <- mlr3resampling::LearnerRegrCVGlmnetSave$new()
  reg.learner.list$cv_glmnet$param_set$values$nfolds <- 3
```

```
}
reg.learner.list # a list of learners.

# 2-fold CV.
kfold <- mlr3::ResamplingCV$new()
kfold$param_set$values$folds <- 2

# Create project grid.
pkg.proj.dir <- tempfile()
pgrid <- mlr3resampling::proj_grid(
  pkg.proj.dir,
  reg.task.list,
  reg.learner.list,
  score_args=mlr3::msrs("regr.rmse"),
  kfold)

## Not run:
test_out <- mlr3resampling::proj_test(pkg.proj.dir)
test_out$learners_history.csv # from AutoTunerTorch_epochs, 2 epochs for testing.
test_out$learners_weights.csv # from LearnerRegrCVGlmnetSave
torch.job.i <- which(pgrid$learner_id=="torch_linear")[1]
mlr3resampling::proj_compute(torch.job.i, pkg.proj.dir)
mlr3resampling::proj_results_save(pkg.proj.dir)
full_out <- mlr3resampling::proj_fread(pkg.proj.dir)
full_out$learners_history.csv # from AutoTunerTorch_epochs, 3 epochs.

## End(Not run)
```

`proj_compute` *Compute resampling results in a project*

Description

Runs `train()` and `predict()`, then a data table with one row is saved to an RDS file in the `grid_jobs` directory.

Usage

```
proj_compute(grid_job_i, proj_dir, verbose=FALSE, process_fun=Sys.getpid)
```

Arguments

`grid_job_i`	integer from 1 to number of jobs (rows in `grid_jobs.csv`).
`proj_dir`	Project directory created by `proj_grid`.
`verbose`	Logical: print messages?
`process_fun`	function called with no arguments (default `Sys.getpid`), should return integer process id number.

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

proj_compute returns a 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(1, pkg.proj.dir)
```

---

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 = save_learner_default, save_pred = FALSE,
  train_seed = 1L, resampling_seed = 1L)
```

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(), typically a list of measures to use for evaluation of predictions on the test set.
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 is to call save_learner method if it exists. 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.
train_seed	integer: random seed to set before training (default 1L for reproducibility, or NA_integer_ to not set seed.
resampling_seed	integer: random seed to set before instantiating each resampling (default 1L).

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(2, pkg.proj.dir)
```

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 one or more CSV files (redundant with RDS data, but more convenient). proj_fread reads the CSV files, adding columns from proj_grid.csv to learners*.csv.

Usage

```
proj_results(proj_dir, verbose=FALSE)
proj_results_save(proj_dir, verbose=FALSE)
proj_fread(proj_dir)
```

Arguments

proj_dir	Project directory created via proj_grid.
verbose	logical: cat progress? (default FALSE)

Details

This is Step 3 out of the typical 3 step pipeline (init grid, submit, read results). Actually, if step 2 worked as intended, then proj_results_save is called at the end of step 2, which saves result files to disk that you can read directly:

results.csv contains test measures for each split.

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.

learners_*.csv only exists if learner column is a named list of data frames: star in file name is expanded using list names, with CSV data taken from atomic columns.

Value

proj_results returns a data table with all columns, whereas proj_results_save returns the same table with only atomic columns. proj_fread returns a list with names corresponding to CSV files in the test directory, and values are the data tables that result from fread.

Author(s)

Toby Dylan Hocking

Examples

```
N <- 80
library(data.table)
set.seed(1)
reg.dt <- data.table(
  x=runif(N, -2, 2),
  noise=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 <- c("x","noise")
  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,
  save_learner=function(L){
    if(inherits(L, "LearnerRegrRpart")){
      list(rpart=L$model$frame)
    }
  },
  order_jobs = function(DT)which(DT$iteration==1)[1:2], # for CRAN.
  score_args=mlr3::msrs(c("regr.rmse", "regr.mae")))
computed <- mlr3resampling::proj_compute_all(pkg.proj.dir)
result_list <- mlr3resampling::proj_fread(pkg.proj.dir)
result_list$learners_rpart.csv # one row per node in decision tree.
result_list$results.csv # test error in regr.* columns.
```

proj_submit *Compute several resampling jobs*

Description

proj_todo determines which jobs remain to be computed. proj_compute_all computes all remaining jobs using future_lapply if available, otherwise lapply. proj_compute_mpi computes all remaining jobs in parallel using MPI (should be run in an R session activated by mpirun or srun). proj_submit is a non-blocking call to SLURM sbatch, asking for a single job with several tasks that run proj_compute_mpi.

Usage

```
proj_todo(proj_dir)
proj_compute_mpi(proj_dir, verbose=FALSE)
proj_compute_all(proj_dir, verbose=FALSE, LAPPLY)
proj_submit(
  proj_dir, tasks = 2, hours = 1, gigabytes = 1,
  verbose = FALSE)
```

Arguments

proj_dir	Project directory created via proj_grid.
tasks	Positive integer: ntasks parameter for SLURM scheduler, one for manager, others are workers.
hours	Hours of walltime to ask the SLURM scheduler.
gigabytes	Gigabytes of memory to ask the SLURM scheduler.
verbose	Logical: print messages?
LAPPLY	Function like lapply (default future.apply::future_lapply if available). Set to lapply for debugging.

Details

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

Value

proj_submit returns the ID of the submitted SLURM job. proj_compute_all and proj_compute_mpi return a data table of results computed. proj_todo returns a vector of job IDs not yet computed.

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")))

## Not run:
if(requireNamespace("future.apply"))future::plan("multisession")
mlr3resampling::proj_compute_all(pkg.proj.dir)
if(requireNamespace("future.apply"))future::plan("sequential")

## End(Not run)
```

`proj_test` *Test a project with smaller data and fewer resampling iterations*

Description

Like testJob, "test" means trying an example with a few small jobs (default one train/test split per algorithm and data set) before running the big calculation with all jobs. Runs proj_grid to create a new project in the test sub-directory, with a smaller number of samples in each task, and with only one iteration per Resampling. Runs proj_compute_all on this new test project, and then reads any CSV result files.

Usage

```
proj_test(
  proj_dir, min_samples_per_stratum = 10,
  edit_learner=edit_learner_default, max_jobs=Inf,
  verbose=FALSE, LAPPLY=NULL)
```

Arguments

proj_dir	Project directory created by proj_grid.
min_samples_per_stratum	Minimum number of samples to include in the smallest stratum. Other strata will be down-sampled proportionally.
edit_learner	Function which inputs a learner object, and changes it to take less time for testing. Default calls edit_learner method if it exists, or for AutoTuner based on LearnerTorch, reduces max epochs and patience to 2.
max_jobs	Numeric, max number of jobs to test (default Inf).
verbose	Logical: print messages?
LAPPLY	Function like lapply (default future.apply::future_lapply if available). Set to lapply for debugging.

Value

Same value as proj_fread on test project (list of data tables).

Author(s)

Toby Dylan Hocking

Examples

```
data.table::setDTthreads(1L)
library(data.table)
N <- 8000
set.seed(1)
reg.dt <- data.table(
  x=runif(N, -2, 2),
  person=factor(rep(c("Alice","Bob"), c(0.1,0.9)*N)))
reg.pattern.list <- list(
  easy=function(x, person)x^2,
  impossible=function(x, person)(x^2)*(-1)^as.integer(person))
```

```
kfold <- mlr3::ResamplingCV$new()
kfold$param_set$values$folds <- 2
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,
  kfold,
  save_learner=function(L){
    if(inherits(L, "LearnerRegrRpart")){
      list(rpart=L$model$frame)
    }
  },
  score_args=mlr3::msrs(c("regr.rmse", "regr.mae")))
mlr3resampling::proj_test(pkg.proj.dir)
```

`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

```
data.table::setDTthreads(1L)
library(data.table)
set.seed(2)
N <- 150L
x <- runif(N, -3, 3)
sim.dt <- data.table(
  x = x,
  y = sin(x) + rnorm(N, sd = 0.5),
  Subset = rep(c("large", "large", "small"), length.out = N))
sim.task <- mlr3::TaskRegr$new("iid_easy", sim.dt, target = "y")
sim.task$col_roles$feature <- "x"
sim.task$col_roles$subset <- "Subset"
soak <- mlr3resampling::ResamplingSameOtherSizesCV$new()
soak$param_set$values$folds <- 5L
learner_list <- list(
  mlr3::lrn("regr.featureless"),
  if(requireNamespace("rpart"))mlr3::lrn("regr.rpart"))
bgrid <- mlr3::benchmark_grid(sim.task, learner_list, soak)
result <- mlr3::benchmark(bgrid)
score.dt <- mlr3resampling::score(result, mlr3::msr("regr.rmse"))
if(requireNamespace("ggplot2")) plot(score.dt)

p.list <- mlr3resampling::pvalue(score.dt)
if(requireNamespace("ggplot2")) plot(p.list)
```

---

pvalue_downsample *P-values for full versus down-sampled SOAK results*

Description

For one test.subset, compute summary statistics and paired two-sided t-test P-values that compare same versus all/other, for two train-size panels: full (n.train.groups == groups) and down-sampled (n.train.groups == min(groups)).

Usage

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

Arguments

score_in	Data table output from score, produced using ResamplingSameOtherSizesCV with down-sampling (typically sizes >= 0). If multiple task_id, test.subset, or algorithm values are present, the first row values are used.
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	Non-negative integer number of digits to print in mean/SD text labels.

Value

List of class "pvalue_downsample" with named elements subset_name, model_name, value.var, stats, and pvalues. subset_name and model_name are inferred from score_in.

Author(s)

Daniel Agyapong

Examples

```
data.table::setDTthreads(1L)
library(data.table)
set.seed(2)
N <- 150L
x <- runif(N, -3, 3)
sim.dt <- data.table(
  x = x,
  y = sin(x) + rnorm(N, sd = 0.5),
  Subset = rep(c("large", "large", "small"), length.out = N))
sim.task <- mlr3::TaskRegr$new("iid_easy", sim.dt, target = "y")
sim.task$col_roles$feature <- "x"
sim.task$col_roles$subset <- "Subset"
soak <- mlr3resampling::ResamplingSameOtherSizesCV$new()
soak$param_set$values$folds <- 5L
soak$param_set$values$sizes <- 0L
learner_list <- list(
  mlr3::lrn("regr.featureless"),
  if(requireNamespace("rpart"))mlr3::lrn("regr.rpart"))
bgrid <- mlr3::benchmark_grid(sim.task, learner_list, soak)
result <- mlr3::benchmark(bgrid)
score.dt <- mlr3resampling::score(result, mlr3::msr("regr.rmse"))
if(requireNamespace("ggplot2")) plot(score.dt)
```

```
down.list <- mlr3resampling::pvalue_downsample(score.dt[
  task_id == "iid_easy" & test.subset == "large" & algorithm == "rpart"])
if(requireNamespace("ggplot2")) plot(down.list)
```

`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 two new roles: `subset` and `fold`.

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).

Stratification

`ResamplingSameOtherSizesCV` supports stratified sampling. The stratification variables are assumed to be discrete, and must be stored in the Task with column role `stratum`. If that role is not set, then all data are assigned to one stratum. In case of multiple stratification variables, each combination of the values of the stratification variables forms a stratum. When `fold` role is set, `stratum` role is not used for fold assignment (folds are taken from `fold` role in that case). When `sizes` param is at least 0, then downsampled train sets are created, and if `stratum` role is set, we attempt to preserve strata proportions in downsampled train sets.

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`. If that role is not set, then each row is assigned to a different group. When `fold` role is set, `group` role is not used for fold assignment (folds are taken from `fold` role in that case). When `sizes` param is at least 0, then downsampled train sets are created, and if `group` role is set, we keep groups together in downsampled train sets.

Subsets

ResamplingSameOtherSizesCV supports fixing a test subset, then training on same/other/all subsets. 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 downsampling should be defined as the seeds parameter, default 1.

The number of downsampling train set sizes should be defined as the sizes parameter, which can also take two special values: default -1 means no downsampling at all, and 0 means only downsampling to the min of same/other sets (in units of groups).

The ratio for downsampling 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 sizes (in units of groups).

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).

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

If fold column role is set, then that column will be used for non-random fold assignment (see Reproducibility vignette). Otherwise, the group_stratum_algo parameter controls the algorithm used for random fold assignment. We want to assign folds such that groups are not split (hard constraint), and strata proportions are preserved (objective to optimize). To see exemples of how this works, read the vignette, Using subset with group and stratum. Note that this feature works on tasks with both stratum and group roles (unlike ResamplingCV). Computing an optimal fold assignment is NP-hard (discrete, need to try all possible assignments of groups to folds to find best solution). Instead of computing the optimal solution (slow), we implement heuristic algorithms (fast), which yield approximate solutions. The choice of which heuristic is controlled by the group_stratum_algo parameter, which can take these values:

"RSS" Default, novel heuristic algorithm which attempts to minimize the residual sum of squares, between actual and ideal values of the strata/fold count matrix. First groups are sorted by RSS (group counts - ideal counts per stratum); ties are broken using number of samples in group, mean(ideal*actual) counts (mean over strata), random group order. Then each group is greedily assigned a fold in order; best fold results in min RSS, ties broken using total counts above ideal. For N data, K folds, G groups, and S strata, the algorithm is O(N + K G S) time and O(N + S K) memory.

"Wasikowski" Same heuristic algorithm as in scikit-learn StratifiedGroupKFold, adapted from https://www.kaggle.com/code/jakubwasikowski/stratified-group-k-fold-cross-validation. First groups are sorted by standard deviation over strata counts; ties are broken by random group order. Then each group is greedily assigned a fold in order; best fold results in the smallest mean SD of the strata/fold count matrix (SD over folds is computed for each stratum, then mean of SDs, minimized over folds), ties broken using number of samples in fold. For N data, K folds, G groups, and S strata, the algorithm is O(N + K^2 G S) time and O(N + S K + S G) memory (could be problematic when G and S are both large).

`"WasikowskiLimitedMemory"` Same logic and time complexity but only `O(N + S K)` memory.

The train/test splits are defined by all possible combinations of test subset, test fold, train subsets (same/other/all), downsampling sizes, and random seeds. After `$instantiate()` is called, `$instance` is a list of data about the splits, with elements:

iteration.dt data table with one row per split.

fold.dt data table with same number of rows as task data.

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
```

`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