Package: mlr3extralearners (via r-universe)

October 11, 2026

Title Extra Learners For mlr3

Version 1.7.0

Description Extra learners for use in mlr3.

License LGPL-3

URL https://mlr3extralearners.mlr-org.com, https://github.com/mlr-org/mlr3extralearners

BugReports https://github.com/mlr-org/mlr3extralearners/issues

Depends mlr3 (>= 1.8.0), R (>= 3.3.0)

Imports checkmate, data.table, methods, mlr3cmprsk (>= 0.0.5), mlr3misc (>= 0.19.0), mlr3proba (>= 0.8.8), paradox (>= 1.0.1), R6, stats, withr

Suggests abess, actuar, ada, adabag, aorsf (>= 0.1.6), apcluster, BART (>= 2.9.10), BiocManager, blockForest (>= 0.2.7), brnn (>= 0.9.4), bst, C50, callr, caret (>= 7.0-1), catboost, class, coin, CoxBoost (>= 1.5.1), crs, Cubist (>= 0.6.0), curl, dbarts, deepnet (>= 0.2.1), earth, elasticnet, evtree, ExhaustiveSearch, fastai (>= 2.2.2), flexsurv (>= 2.3.2), FNN, formattable, frbs (>= 3.2-0), future, gbm (>= 2.2.3), glmnet (>= 5.0), grf (>= 2.0.0), GPfit (>= 1.0-9), h2o (>= 3.44.0.3), jsonlite, keras (>= 2.3.0), kernlab, klaR, knitr, laGP (>= 1.5-9), lgr, LiblineaR, lightgbm (>= 4.5.0), lme4 (>= 1.1-38), MASS, mboost (>= 2.9-11), mda, mgcv (>= 1.9-4), mirai, mlr3cluster, mlr3learners (>= 0.12.0), mlr3pipelines, mvtnorm, ncvreg (>= 3.16.0), neuralnet, nnet, partykit (>= 1.2-21), penalized (>= 0.9-53), pls (>= 2.9-0), polyreg, prioritylasso (>= 0.4.0), qgam, randomForest, randomForestSRC (>= 3.6.2), ranger (>= 0.18.0), remotes, reticulate, rFerns, rJava, rmarkdown, ropls, rotationForest, rpart, RRF (>= 1.9.4.1), rsm, RSNNS (>= 0.4-17), rvest, RWeka, sandwich, sda (>= 1.3.9), sparsediscrim, sparseLDA (>= 0.1-9), stepPlr, survdistr (>= 0.0.3), survdnn (>= 0.7.6), survival, survivalsvm (>= 0.0.6), tensorflow (>= 2.0.0), testthat, tgp (>= 2.4-23), torch (>= 0.16.3), xgboost (>= 3.2.1.1)

VignetteBuilder knitr, rmarkdown

Remotes bioc::ropls, catboost/catboost/catboost/R-package, mlr-org/mlr3cmprsk, mlr-org/mlr3proba

Config/Needs/website rmarkdown

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Config/testthat/parallel false

Encoding UTF-8

NeedsCompilation no

Roxygen list(markdown = TRUE, r6 = TRUE)

Config/pak/sysreqs cmake make libicu-dev libuv1-dev

Repository https://mlr-org.r-universe.dev

Date/Publication 2026-09-15 15:32:13 UTC

RemoteUrl https://github.com/mlr-org/mlr3extralearners

RemoteRef v1.7.0

RemoteSha c512a293fe6d06af07898b0a742e446bdd23c5e8

Contents

mlr3extralearners-package . . . 6
assert_python_packages . . . 7
install_learners . . . 8
list_mlr3learners . . . 8
mlr_learners_classif.abess . . . 9
mlr_learners_classif.ada . . . 11
mlr_learners_classif.adabag . . . 14
mlr_learners_classif.AdaBoostM1 . . . 17
mlr_learners_classif.aorsf . . . 20
mlr_learners_classif.bam . . . 23
mlr_learners_classif.bart . . . 26
mlr_learners_classif.bayes_net . . . 29
mlr_learners_classif.blockforest . . . 32
mlr_learners_classif.bst . . . 35
mlr_learners_classif.C50 . . . 38
mlr_learners_classif.catboost . . . 39
mlr_learners_classif.cforest . . . 44
mlr_learners_classif.ctree . . . 47
mlr_learners_classif.cv_ncvreg . . . 50
mlr_learners_classif.dbnDNN . . . 53
mlr_learners_classif.decision_stump . . . 55
mlr_learners_classif.decision_table . . . 58
mlr_learners_classif.dlda . . . 62
mlr_learners_classif.earth . . . 64

mlr_learners_classif.evtree . . . 67
mlr_learners_classif.exhaustive_search . . . 69
mlr_learners_classif.fastai . . . 71
mlr_learners_classif.fnn . . . 75
mlr_learners_classif.gam . . . 77
mlr_learners_classif.gamboost . . . 79
mlr_learners_classif.gausspr . . . 82
mlr_learners_classif.gbm . . . 84
mlr_learners_classif.glmboost . . . 87
mlr_learners_classif.glmer . . . 89
mlr_learners_classif.h2o.deeplearning . . . 92
mlr_learners_classif.h2o.gbm . . . 95
mlr_learners_classif.h2o.glm . . . 98
mlr_learners_classif.h2o.randomForest . . . 102
mlr_learners_classif.IBk . . . 105
mlr_learners_classif.imbalanced_rfsrc . . . 108
mlr_learners_classif.J48 . . . 112
mlr_learners_classif.JRip . . . 115
mlr_learners_classif.knn . . . 118
mlr_learners_classif.kstar . . . 120
mlr_learners_classif.ksvm . . . 123
mlr_learners_classif.liblinear . . . 125
mlr_learners_classif.lightgbm . . . 128
mlr_learners_classif.LMT . . . 134
mlr_learners_classif.logistic . . . 137
mlr_learners_classif.lssvm . . . 140
mlr_learners_classif.lvq1 . . . 142
mlr_learners_classif.mda . . . 145
mlr_learners_classif.mdeb . . . 147
mlr_learners_classif.mlp_rsnns . . . 149
mlr_learners_classif.mob . . . 152
mlr_learners_classif.multilayer_perceptron . . . 155
mlr_learners_classif.naive_bayes_multinomial . . . 158
mlr_learners_classif.naive_bayes_weka . . . 161
mlr_learners_classif.neuralnet . . . 164
mlr_learners_classif.nnTrain . . . 167
mlr_learners_classif.OneR . . . 169
mlr_learners_classif.opls . . . 172
mlr_learners_classif.PART . . . 175
mlr_learners_classif.penalized . . . 178
mlr_learners_classif.plsdaCaret . . . 181
mlr_learners_classif.polyFit . . . 183
mlr_learners_classif.priority_lasso . . . 184
mlr_learners_classif.random_forest_weka . . . 188
mlr_learners_classif.random_tree . . . 191
mlr_learners_classif.randomForest . . . 194
mlr_learners_classif.rda . . . 197
mlr_learners_classif.reptree . . . 200

mlr_learners_classif.rFerns . . . 203
mlr_learners_classif.rfsrc . . . 205
mlr_learners_classif.rotationForest . . . 209
mlr_learners_classif.saeDNN . . . 211
mlr_learners_classif.sda . . . 213
mlr_learners_classif.sdlda . . . 216
mlr_learners_classif.sgd . . . 218
mlr_learners_classif.simple_logistic . . . 221
mlr_learners_classif.smo . . . 224
mlr_learners_classif.sparseLDA . . . 228
mlr_learners_classif.stepPlr . . . 230
mlr_learners_classif.tabfm . . . 232
mlr_learners_classif.tabpfn . . . 236
mlr_learners_classif.voted_perceptron . . . 240
mlr_learners_cmprsk.coxboost . . . 243
mlr_learners_cmprsk.rfsrc . . . 246
mlr_learners_regr.abess . . . 250
mlr_learners_regr.aorsf . . . 253
mlr_learners_regr.bam . . . 256
mlr_learners_regr.bart . . . 259
mlr_learners_regr.bcart . . . 262
mlr_learners_regr.bgp . . . 265
mlr_learners_regr.bgpllm . . . 267
mlr_learners_regr.blm . . . 270
mlr_learners_regr.blockforest . . . 272
mlr_learners_regr.botorch_fullybayesian . . . 275
mlr_learners_regr.botorch_mixedsingletaskgp . . . 277
mlr_learners_regr.botorch_singletaskgp . . . 279
mlr_learners_regr.brnn . . . 281
mlr_learners_regr.bst . . . 283
mlr_learners_regr.btgp . . . 285
mlr_learners_regr.btgpllm . . . 288
mlr_learners_regr.btlm . . . 291
mlr_learners_regr.catboost . . . 294
mlr_learners_regr.cforest . . . 298
mlr_learners_regr.crs . . . 302
mlr_learners_regr.ctree . . . 304
mlr_learners_regr.cubist . . . 307
mlr_learners_regr.cv_ncvreg . . . 310
mlr_learners_regr.decision_stump . . . 312
mlr_learners_regr.decision_table . . . 315
mlr_learners_regr.earth . . . 319
mlr_learners_regr.evtree . . . 322
mlr_learners_regr.exhaustive_search . . . 323
mlr_learners_regr.fnn . . . 326
mlr_learners_regr.frbs . . . 328
mlr_learners_regr.gam . . . 330
mlr_learners_regr.gamboost . . . 333

mlr_learners_regr.gaussian_processes . . . 335
mlr_learners_regr.gausspr . . . 338
mlr_learners_regr.gbm . . . 340
mlr_learners_regr.glm . . . 343
mlr_learners_regr.glmboost . . . 345
mlr_learners_regr.gpfit . . . 348
mlr_learners_regr.grf . . . 350
mlr_learners_regr.h2o.deeplearning . . . 353
mlr_learners_regr.h2o.gbm . . . 356
mlr_learners_regr.h2o.glm . . . 359
mlr_learners_regr.h2o.randomForest . . . 362
mlr_learners_regr.IBk . . . 365
mlr_learners_regr.kstar . . . 368
mlr_learners_regr.ksvm . . . 371
mlr_learners_regr.laGP . . . 374
mlr_learners_regr.liblinear . . . 376
mlr_learners_regr.lightgbm . . . 379
mlr_learners_regr.linear_regression . . . 384
mlr_learners_regr.lmer . . . 387
mlr_learners_regr.m5p . . . 390
mlr_learners_regr.M5Rules . . . 393
mlr_learners_regr.mars . . . 396
mlr_learners_regr.mob . . . 398
mlr_learners_regr.mqgam . . . 401
mlr_learners_regr.multilayer_perceptron . . . 403
mlr_learners_regr.pcr . . . 406
mlr_learners_regr.penalized . . . 409
mlr_learners_regr.plsr . . . 411
mlr_learners_regr.polyFit . . . 413
mlr_learners_regr.polynomial . . . 415
mlr_learners_regr.priority_lasso . . . 417
mlr_learners_regr.qgam . . . 421
mlr_learners_regr.random_forest_weka . . . 423
mlr_learners_regr.random_tree . . . 426
mlr_learners_regr.randomForest . . . 429
mlr_learners_regr.reptree . . . 432
mlr_learners_regr.rfsrc . . . 435
mlr_learners_regr.RRF . . . 439
mlr_learners_regr.rsm . . . 441
mlr_learners_regr.rvm . . . 444
mlr_learners_regr.sgd . . . 446
mlr_learners_regr.simple_linear_regression . . . 449
mlr_learners_regr.smo_reg . . . 452
mlr_learners_regr.tabfm . . . 456
mlr_learners_regr.tabpfn . . . 460
mlr_learners_surv.aorsf . . . 463
mlr_learners_surv.bart . . . 467
mlr_learners_surv.blackboost . . . 470

mlr_learners_surv.blockforest . . . 474
mlr_learners_surv.cforest . . . 477
mlr_learners_surv.coxboost . . . 480
mlr_learners_surv.ctree . . . 483
mlr_learners_surv.cv_coxboost . . . 486
mlr_learners_surv.cv_glmnet . . . 489
mlr_learners_surv.cv_ncvsurv . . . 493
mlr_learners_surv.flexreg . . . 496
mlr_learners_surv.flexspline . . . 499
mlr_learners_surv.gam.cox . . . 502
mlr_learners_surv.gamboost . . . 504
mlr_learners_surv.gbm . . . 507
mlr_learners_surv.glmboost . . . 510
mlr_learners_surv.glmnet . . . 513
mlr_learners_surv.mboost . . . 518
mlr_learners_surv.nelson . . . 520
mlr_learners_surv.penalized . . . 523
mlr_learners_surv.priority_lasso . . . 526
mlr_learners_surv.ranger . . . 530
mlr_learners_surv.rfsrc . . . 533
mlr_learners_surv.survdnn . . . 537
mlr_learners_surv.svm . . . 540
mlr_learners_surv.xgboost.aft . . . 543
mlr_learners_surv.xgboost.cox . . . 547

Index **552**

`mlr3extralearners-package`

mlr3extralearners: Extra Learners For mlr3

Description

Extra learners for use in mlr3.

Author(s)

Maintainer: Marc Becker <marcbecker@posteo.de> (ORCID)

Authors:

- Marc Becker <marcbecker@posteo.de> (ORCID)
- Raphael Sonabend <raphaelsonabend@gmail.com> (ORCID)
- Patrick Schratz <patrick.schratz@gmail.com> (ORCID)
- Sebastian Fischer <sebf.fischer@gmail.com>
- John Zobolas <bblodfon@gmail.com> (ORCID)

Other contributors:

- Lorenz A. Kapsner <lorenz.kapsner@gmail.com> (ORCID) [contributor]
- Lennart Schneider <lennart.sch@web.de> (ORCID) [contributor]
- Stephen A Lauer <stephenalauer@gmail.com> (ORCID) [contributor]
- Pierre Camilleri <camilleri_pierre@hotmail.fr> [contributor]
- Javier García <geshter@hotmail.com> [contributor]
- Keenan Ganz <ganzkeenan1@gmail.com> [contributor]
- Byron Jaeger <bjaeger@wakehealth.edu> [contributor]
- Zezhi Wang <homura@mail.ustc.edu.cn> [contributor]
- Lukas Burk <burk@leibniz-bips.de> (ORCID) [contributor]
- Lona Koers <lona.koers@gmail.com> [contributor]
- Anna Nazarova <anna.nazarova@outlook.com> [contributor]
- Baisu Zhou <baisu.zhou@outlook.com> [contributor]
- Nikolai German <niko.german@gmail.com> [contributor]
- Alexander Winterstetter <alexanderwinterstetter@gmail.com> [contributor]
- Imad El Badisy <elbadisyimad@gmail.com> [contributor]

See Also

Useful links:

- https://mlr3extralearners.mlr-org.com
- https://github.com/mlr-org/mlr3extralearners
- Report bugs at https://github.com/mlr-org/mlr3extralearners/issues

---

`assert_python_packages` *Assert Python Packages*

Description

Assert that the given Python packages are available.

Usage

```
assert_python_packages(packages, python_version = NULL)
```

Arguments

`packages`	(`character()`) Python packages to check.
`python_version`	(`character(1)`) Python version to use. If `NULL`, the default Python version is used.

Value

(character())

---

install_learners *Install Learner Dependencies*

Description

Install required dependencies for specified learners. Works for packages on CRAN, Bioconductor and GitHub, otherwise must be manually installed.

Usage

```
install_learners(.keys, repos = "https://cloud.r-project.org", ...)
```

Arguments

.keys	(character()) Keys passed to mlr_learners specifying learners to install.
repos	(character(1)) Passed to utils::install.packages.
...	(ANY) Additional options to pass to utils::install.packages(), BiocManager::install() or remotes::install_github().

---

list_mlr3learners *List Learners in mlr3verse*

Description

Lists all learners, properties, and associated packages in a table that can be filtered and queried.

Usage

```
list_mlr3learners(select = NULL, filter = NULL)
```

Arguments

select	character() Passed to data.table::subset.
filter	list() Named list of conditions to filter on, names correspond to column names in table.

`mlr_learners_classif.abess`
Classification Abess Learner

Description

Adaptive best-subset selection for classification. Calls `abess::abess()` from **abess**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.abess")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **abess**

Parameters

Id	Type	Default	Levels	Range
family	character	-	binomial, multinomial, ordinal	-
tune.path	character	sequence	sequence, gsection	-
tune.type	character	gic	gic, aic, bic, ebic, cv	-
normalize	integer	NULL		$(-\infty, \infty)$
support.size	untyped	NULL		-
c.max	integer	2		$[1, \infty)$
gs.range	untyped	NULL		-
lambda	numeric	0		$[0, \infty)$
always.include	untyped	NULL		-
group.index	untyped	NULL		-
init.active.set	untyped	NULL		-
splicing.type	integer	2		$[1, 2]$
max.splicing.iter	integer	20		$[1, \infty)$
screening.num	integer	NULL		$[0, \infty)$
important.search	integer	NULL		$[0, \infty)$
warm.start	logical	TRUE	TRUE, FALSE	-
nfolds	integer	5		$(-\infty, \infty)$
foldid	untyped	NULL		-
cov.update	logical	FALSE	TRUE, FALSE	-
newton	character	exact	exact, approx	-
newton.thresh	numeric	1e-06		$[0, \infty)$

max.newton.iter	integer	NULL		$[1, \infty)$
early.stop	logical	FALSE	TRUE, FALSE	-
ic.scale	numeric	1		$[0, \infty)$
num.threads	integer	0		$[0, \infty)$
seed	integer	0		$(-\infty, \infty)$

Initial parameter values

- `num.threads`: This parameter is initialized to 1 (default is 0) to avoid conflicts with the mlr3 parallelization.
- `family`: Depends on the task type, if the parameter `family` is `NULL`, it is set to `"binomial"` for binary classification tasks and to `"multinomial"` for multiclass classification problems.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifAbess`

Methods

Public methods:

- `LearnerClassifAbess$new()`
- `LearnerClassifAbess$selected_features()`
- `LearnerClassifAbess$clone()`

`LearnerClassifAbess$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifAbess$new()
```

`LearnerClassifAbess$selected_features()`: Extract the name of selected features from the model by `abess::extract()`.

Usage:

```
LearnerClassifAbess$selected_features()
```

Returns: The names of selected features

`LearnerClassifAbess$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifAbess$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bbayukari

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.abess")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.ada`

Classification Boosting Learner

Description

Classification using ada. Calls `ada::ada()` from **ada**.

Initial parameter values

- `xval` is set to `0` by default for speed.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.ada")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **ada**, **rpart**

Parameters

Id	Type	Default	Levels	Range
bag.frac	numeric	0.5		$[0, 1]$
bag.shift	logical	FALSE	TRUE, FALSE	-
cp	numeric	0.01		$[0, 1]$
delta	numeric	1e-10		$[0, \infty)$
iter	integer	50		$[1, \infty)$
loss	character	exponential	exponential, logistic	-
max.iter	integer	20		$[1, \infty)$
maxcompete	integer	4		$[0, \infty)$
maxdepth	integer	30		$[1, 30]$
maxsurrogate	integer	5		$[0, \infty)$
minbucket	integer	-		$[1, \infty)$
minsplit	integer	20		$[1, \infty)$
model.coef	logical	TRUE	TRUE, FALSE	-
n.iter	integer	50		$[1, \infty)$
nu	numeric	0.1		$[0, \infty)$
surrogatestyle	integer	0		$[0, 1]$
type	character	discrete	discrete, real, gentle	-
usesurrogate	integer	2		$[0, 2]$
verbose	logical	FALSE	TRUE, FALSE	-
xval	integer	0		$[0, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifAdaBoosting`

Methods

Public methods:

- `LearnerClassifAdaBoosting$new()`
- `LearnerClassifAdaBoosting$oob_error()`
- `LearnerClassifAdaBoosting$clone()`

`LearnerClassifAdaBoosting$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifAdaBoosting$new()
```

`LearnerClassifAdaBoosting$oob_error()`: The oob error is extracted extracted from the model slot `$model$errs`.

Usage:

```
LearnerClassifAdaBoosting$oob_error()
```

Returns: Named `numeric()`.

`LearnerClassifAdaBoosting$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifAdaBoosting$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.ada")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.adabag`

Classification Boosting Learner

Description

Classification boosting algorithm. Calls `adabag::boosting()` from **adabag**.

Initial parameter values

- `xval`:
  - Actual default: 10L
  - Initial value: 0L
  - Reason for change: Set to 0 for speed.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.adabag")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **adabag**, **rpart**

Parameters

Id	Type	Default	Levels	Range
boos	logical	TRUE	TRUE, FALSE	-
coeflearn	character	Breiman	Breiman, Freund, Zhu	-
cp	numeric	0.01		$[0, 1]$
maxcompete	integer	4		$[0, \infty)$
maxdepth	integer	30		$[1, 30]$
maxsurrogate	integer	5		$[0, \infty)$
mfinal	integer	100		$[1, \infty)$
minbucket	integer	-		$[1, \infty)$
minsplit	integer	20		$[1, \infty)$
newmfinal	integer	-		$(-\infty, \infty)$
surrogatestyle	integer	0		$[0, 1]$
usesurrogate	integer	2		$[0, 2]$
xval	integer	0		$[0, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifAdabag`

Methods

Public methods:

- `LearnerClassifAdabag$new()`
- `LearnerClassifAdabag$importance()`
- `LearnerClassifAdabag$clone()`

`LearnerClassifAdabag$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifAdabag$new()
```

`LearnerClassifAdabag$importance()`: The importance scores are extracted from the model.

Usage:

```
LearnerClassifAdabag$importance()
```

Returns: Named `numeric()`.

`LearnerClassifAdabag$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifAdabag$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Alfaro, Esteban, Gamez, Matias, García, Noelia (2013). "adabag: An R Package for Classification with Boosting and Bagging." *Journal of Statistical Software*, **54**(2), 1-35. doi:10.18637/jss.v054.i02. https://www.jstatsoft.org/index.php/jss/article/view/v054i02.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.adabag", mfinal = 10L)
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
```

```
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.AdaBoostM1`
Classification AdaBoostM1 Learner

Description

Adaptive boosting algorithm for classification. Calls `RWeka::AdaBoostM1()` from **RWeka**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.AdaBoostM1")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
P	integer	100		$[90, 100]$
Q	logical	FALSE	TRUE, FALSE	-
S	integer	1		$[1, \infty)$
I	integer	10		$[1, \infty)$
W	untyped	"DecisionStump"		-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Parameter changes

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifAdaBoostM1`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifAdaBoostM1$new()`
- `LearnerClassifAdaBoostM1$marshal()`
- `LearnerClassifAdaBoostM1$unmarshal()`
- `LearnerClassifAdaBoostM1$clone()`

`LearnerClassifAdaBoostM1$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifAdaBoostM1$new()
```

`LearnerClassifAdaBoostM1$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifAdaBoostM1$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifAdaBoostM1$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifAdaBoostM1$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifAdaBoostM1$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifAdaBoostM1$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

henrifnk

References

Freund, Yoav, Schapire, E R, others (1996). "Experiments with a new boosting algorithm." In *icml*, volume 96, 148–156. Citeseer.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.AdaBoostM1")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.aorsf`

Oblique Random Forest Classifier

Description

Accelerated oblique random classification forest. Calls `aorsf::orsf()` from **aorsf**. Note that although the learner has the property `"missing"` and it can in principle deal with missing values, the behavior has to be configured using the parameter `na_action`.

Initial parameter values

- `n_thread`: This parameter is initialized to 1 (default is 0) to avoid conflicts with the mlr3 parallelization.
- `pred_simplify` has to be TRUE, otherwise response is NA in prediction

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.aorsf")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **aorsf**

Parameters

Id	Type	Default	Levels	Range
attach_data	logical	TRUE	TRUE, FALSE	-
epsilon	numeric	1e-09		$[0, \infty)$
importance	character	anova	none, anova, negate, permute	-
importance_max_pvalue	numeric	0.01		$[1e-04, 0.9999]$

leaf_min_events	integer	1		$[1, \infty)$
leaf_min_obs	integer	5		$[1, \infty)$
max_iter	integer	20		$[1, \infty)$
method	character	glm	glm, net, pca, random	-
mtry	integer	NULL		$[1, \infty)$
mtry_ratio	numeric	-		$[0, 1]$
n_retry	integer	3		$[0, \infty)$
n_split	integer	5		$[1, \infty)$
n_thread	integer	-		$[0, \infty)$
n_tree	integer	500		$[1, \infty)$
na_action	character	fail	fail, impute_meanmode	-
net_mix	numeric	0.5		$(-\infty, \infty)$
oobag	logical	FALSE	TRUE, FALSE	-
oobag_eval_every	integer	NULL		$[1, \infty)$
oobag_fun	untyped	NULL		-
oobag_pred_type	character	prob	none, leaf, prob, class	-
pred_aggregate	logical	TRUE	TRUE, FALSE	-
sample_fraction	numeric	0.632		$[0, 1]$
sample_with_replacement	logical	TRUE	TRUE, FALSE	-
scale_x	logical	FALSE	TRUE, FALSE	-
split_min_events	integer	5		$[1, \infty)$
split_min_obs	integer	10		$[1, \infty)$
split_min_stat	numeric	NULL		$[0, \infty)$
split_rule	character	gini	gini, cstat	-
target_df	integer	NULL		$[1, \infty)$
tree_seeds	integer	NULL		$[1, \infty)$
verbose_progress	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifObliqueRandomForest`

Methods

Public methods:

- `LearnerClassifObliqueRandomForest$new()`
- `LearnerClassifObliqueRandomForest$oob_error()`
- `LearnerClassifObliqueRandomForest$importance()`
- `LearnerClassifObliqueRandomForest$clone()`

`LearnerClassifObliqueRandomForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifObliqueRandomForest$new()
```

`LearnerClassifObliqueRandomForest$oob_error()`: OOB concordance error extracted from the model slot `eval_oobag$stat_values`

Usage:

```
LearnerClassifObliqueRandomForest$oob_error()
```

Returns: `numeric()`.

`LearnerClassifObliqueRandomForest$importance()`: The importance scores are extracted from the model.

Usage:

```
LearnerClassifObliqueRandomForest$importance()
```

Returns: Named `numeric()`.

`LearnerClassifObliqueRandomForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifObliqueRandomForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.aorsf")
print(learner)

# Define a Task
task = tsk("sonar")
```

```
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.bam`

Classification Fast Generalized Additive Model Learner

Description

Fast generalized additive models for large datasets. Calls `mgcv::bam()` from package **mgcv** with a `binomial` family. For smaller datasets, use `classif.gam` instead.

Formula

A gam formula specific to the task at hand is required for the `formula` parameter (see example and `?mgcv::formula.gam`). Beware, if no formula is provided, a fallback formula is used that will make the model behave like a glm (this behavior is required for the unit tests). Only features specified in the formula will be used, superseding columns with col_roles "feature" in the task.

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated during training via the `offset` argument in `mgcv::bam()`. No offset is applied during prediction for this learner.

Custom mlr3 parameters

- `family` is always `binomial`, so it is not exposed as a hyperparameter.
- `control` is passed to `mgcv::gam.control()` as a list. Unlike `classif.gam`, the individual control options are not exposed as hyperparameters, because `nthreads` is both a control option and an argument of `mgcv::bam()`.
- `discrete` is a parameter of both `mgcv::bam()` and `mgcv::predict.bam()`, with different defaults. Only the training parameter is exposed, and prediction uses the default of `mgcv::predict.bam()`.
- Only the training parameter `nthreads` is tagged with `"threads"`, so `mlr3::set_threads()` does not change the prediction parameter `n.threads`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.bam")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3extralearners**, **mgcv**

Parameters

Id	Type	Default	Levels	Range
formula	untyped	-		-
method	character	fREML	fREML, GCV.Cp, GACV.Cp, REML, P-REML, ML, P-ML	-
control	untyped	list()		-
select	logical	FALSE	TRUE, FALSE	-
scale	numeric	0		$(-\infty, \infty)$
gamma	numeric	1		$[1, \infty)$
knots	untyped	NULL		-
sp	untyped	NULL		-
min.sp	untyped	NULL		-
paraPen	untyped	NULL		-
chunk.size	integer	10000		$[1, \infty)$
rho	numeric	0		$(-\infty, \infty)$
AR.start	untyped	NULL		-
discrete	logical	FALSE	TRUE, FALSE	-
nthreads	integer	1		$[1, \infty)$
use.chol	logical	FALSE	TRUE, FALSE	-
samfrac	numeric	1		$[0, 1]$
coef	untyped	NULL		-
drop.unused.levels	logical	TRUE	TRUE, FALSE	-
G	untyped	NULL		-
drop.intercept	logical	NULL	TRUE, FALSE	-
in.out	untyped	NULL		-
nei	untyped	NULL		-
cluster	untyped	NULL		-
gc.level	integer	0		$[0, \infty)$
terms	untyped	NULL		-
exclude	untyped	NULL		-
block.size	integer	50000		$(-\infty, \infty)$
n.threads	integer	1		$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifBam`

Methods

Public methods:

- `LearnerClassifBam$new()`
- `LearnerClassifBam$clone()`

`LearnerClassifBam$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifBam$new()
```

`LearnerClassifBam$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifBam$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Tino-Rg

References

Hastie, J T, Tibshirani, J R (2017). *Generalized additive models*. Routledge.

Wood, Simon (2012). "mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML smoothness estimation."

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# simple example
t = tsk("spam")$filter(1:1000)
l = lrn("classif.bam")
l$param_set$set_values(
  formula = type ~ s(george, k = 3) + s(charDollar, k = 3) + s(edu)
)
l$train(t)
l$model
```

`mlr_learners_classif.bart`

Classification BART (Bayesian Additive Regression Trees) Learner

---

Description

Bayesian Additive Regression Trees are similar to gradient boosting algorithms. The classification problem is solved by 0-1 encoding of the two-class targets and setting the decision threshold to p = 0.5 during the prediction phase. Calls `dbarts::bart()` from **dbarts**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.bart")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **dbarts**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	200		$[1, \infty)$
k	numeric	2		$[0, \infty)$
power	numeric	2		$[0, \infty)$
base	numeric	0.95		$[0, 1]$
binaryOffset	numeric	0		$(-\infty, \infty)$
ndpost	integer	1000		$[1, \infty)$
nskip	integer	100		$[0, \infty)$
printevery	integer	100		$[0, \infty)$
keepevery	integer	1		$[1, \infty)$

keeptrainfits	logical	TRUE	TRUE, FALSE	-
usequants	logical	FALSE	TRUE, FALSE	-
numcut	integer	100		$[1, \infty)$
printcutoffs	integer	0		$(-\infty, \infty)$
verbose	logical	FALSE	TRUE, FALSE	-
nthread	integer	1		$(-\infty, \infty)$
keepcall	logical	TRUE	TRUE, FALSE	-
sampleronly	logical	FALSE	TRUE, FALSE	-
seed	integer	NA		$(-\infty, \infty)$
proposalprobs	untyped	NULL		-
splitprobs	untyped	NULL		-
keepsampler	logical	-	TRUE, FALSE	-
n.threads	integer	-		$(-\infty, \infty)$

Parameter Changes

- Parameter: keeptrees
- Original: FALSE
- New: TRUE
- Reason: Required for prediction
- Parameter: offset
- The parameter is removed, because only `dbarts::bart2` allows an offset during training, and therefore the offset parameter in `dbarts:::predict.bart` is irrelevant for `dbarts::dbart`.
- Parameter: nchain, combineChains, combinechains
- The parameters are removed as parallelization of multiple models is handled by future.
- Parameter: sigest, sigdf, sigquant, keeptrees
- Regression only.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifBart`

Methods

Public methods:

- `LearnerClassifBart$new()`
- `LearnerClassifBart$clone()`

`LearnerClassifBart$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifBart$new()
```

`LearnerClassifBart$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifBart$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ck37

References

Sparapani, Rodney, Spanbauer, Charles, McCulloch, Robert (2021). "Nonparametric machine learning and efficient computation with bayesian additive regression trees: the BART R package." *Journal of Statistical Software*, **97**, 1–66.

Chipman, A H, George, I E, McCulloch, E R (2010). "BART: Bayesian additive regression trees." *The Annals of Applied Statistics*, **4**(1), 266–298.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.bart")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.bayes_net`
Classification Bayes Network Learner

Description

Bayes Network learning using various search algorithms. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern
- `N` removed:
  - Initial structure is empty
- `P` removed:
  - Maximum number of parents
- `R` removed:
  - Random order
- `mbc` removed:
  - Applies a Markov Blanket correction to the network structure, after a network structure is learned
- `S` removed:
  - Score type
- `A` removed:
  - Initial count (alpha)
- Reason for change: The parameters are removed because they don't work out of the box and it's unclear how to use them.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.bayes_net")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels
subset	untyped	-	
na.action	untyped	-	
D	logical	-	TRUE, FALSE
B	untyped	-	
Q	character	-	global.K2, global.HillClimber, global.SimulatedAnnealing, global.TabuSe
E	character	-	estimate.SimpleEstimator, estimate.BMAEstimator, estimate.MultiNomia
output_debug_info	logical	FALSE	TRUE, FALSE
do_not_check_capabilities	logical	FALSE	TRUE, FALSE
num_decimal_places	integer	2	
batch_size	integer	100	
options	untyped	NULL	

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifBayesNet`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifBayesNet$new()`
- `LearnerClassifBayesNet$marshal()`
- `LearnerClassifBayesNet$unmarshal()`
- `LearnerClassifBayesNet$clone()`

`LearnerClassifBayesNet$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifBayesNet$new()
```

`LearnerClassifBayesNet$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifBayesNet$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifBayesNet$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifBayesNet$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifBayesNet$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifBayesNet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.bayes_net")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.blockforest`
BlockForest Classification Learner

Description

Random forests for blocks of clinical and omics covariate data. Calls `blockForest::blockfor()` from package **blockForest**.

In this learner, only the trained forest object (`$forest`) is retained. The optimized block-specific tuning parameters (`paramvalues`) and the biased OOB error estimate (`biased_oob_error_donotuse`) are discarded, as they are either not needed for downstream use or not reliable for performance estimation.

Initial parameter values

- `num.threads` is initialized to 1 to avoid conflicts with parallelization via **future**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.blockforest")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **blockForest**

Parameters

Id	Type	Default	Levels	Range
blocks	untyped	-		-
block.method	character	BlockForest	BlockForest, RandomBlock, BlockVarSel, VarProb, SplitWeights	-
num.trees	integer	2000		$[1, \infty)$
mtry	untyped	NULL		-
nsets	integer	300		$[1, \infty)$
num.trees.pre	integer	1500		$[1, \infty)$
splitrule	character	extratrees	extratrees, gini	-
always.select.block	integer	0		$[0, 1]$
importance	character	-	none, impurity, impurity_corrected, permutation	-
num.threads	integer	-		$[1, \infty)$
seed	integer	NULL		$(-\infty, \infty$
verbose	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifBlockForest`

Methods

Public methods:

- `LearnerClassifBlockForest$new()`
- `LearnerClassifBlockForest$importance()`
- `LearnerClassifBlockForest$clone()`

`LearnerClassifBlockForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifBlockForest$new()
```

`LearnerClassifBlockForest$importance()`: The importance scores are extracted from the model slot `variable.importance`.

Usage:

```
LearnerClassifBlockForest$importance()
```

Returns: Named `numeric()`.

`LearnerClassifBlockForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifBlockForest$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

bblodfon

References

Hornung, R., Wright, N. M (2019). "Block Forests: Random forests for blocks of clinical and omics covariate data." *BMC Bioinformatics*, **20**(1), 1–17. doi:10.1186/s128590192942y. https://doi.org/10.1186/s12859-019-2942-y.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# check task's features
task$feature_names

# partition features to 2 blocks
blocks = list(bl1 = 1:42, bl2 = 43:60)

# define learner
learner = lrn("classif.blockforest", blocks = blocks,
              importance = "permutation", nsets = 10, predict_type = "prob",
              num.trees = 50, num.trees.pre = 10, splitrule = "gini")
```

```
# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# feature importance
learner$importance()

# Make predictions for the test observations
pred = learner$predict(task, row_ids = ids$test)
pred

# Score the predictions
pred$score()
```

`mlr_learners_classif.bst`

Classification Gradient Boosting Learner

Description

Classification gradient boosting learner. Calls `bst::bst()` from **bst**.

Initial parameter values

- `Learner` = `"ls"`: Default base learner type
- `xval` = `0`: No cross-validation
- `maxdepth` = `1`: Maximum tree depth

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.bst")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “numeric”
- Required Packages: **mlr3**, **mlr3extralearners**, **bst**, **rpart**

Parameters

Id	Type	Default	Levels
center	logical	FALSE	TRUE, FALSE
coefir	untyped	NULL	
cost	numeric	0.5	
cp	numeric	0.01	
df	integer	4	
family	character	hinge	hinge, hinge2, binom, thingeDC, tbinomDC, binomdDC, loss, clossR, clossRMM, clo
f.init	untyped	NULL	
fk	untyped	NULL	
intercept	logical	TRUE	TRUE, FALSE
iter	integer	1	
Learner	character	ls	ls, sm, tree
maxdepth	integer	1	
maxsurrogate	integer	5	
minbucket	integer	-	
minsplit	integer	20	
mstop	integer	50	
numsample	integer	50	
nu	numeric	0.1	
q	numeric	-	
qh	numeric	-	
s	numeric	-	
sh	numeric	-	
start	logical	FALSE	TRUE, FALSE
surrogatestyle	integer	0	
threshold	character	adaptive	adaptive, fixed
trace	logical	FALSE	TRUE, FALSE
trun	logical	FALSE	TRUE, FALSE
twinboost	logical	FALSE	TRUE, FALSE
twintype	integer	1	
xselect.init	untyped	NULL	
xval	integer	10	

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifBst`

Methods

Public methods:

- `LearnerClassifBst$new()`
- `LearnerClassifBst$clone()`

`LearnerClassifBst$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifBst$new()
```

`LearnerClassifBst$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifBst$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

Only classification-appropriate loss functions are available for the `family` parameter.

Author(s)

annanzrv

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.bst")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.C50`
Classification C5.0 Learner

Description

Decision Tree Algorithm. Calls `C50::C5.0.formula()` from **C50**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.C50")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **C50**

Parameters

Id	Type	Default	Levels	Range
trials	integer	1		$[1, \infty)$
rules	logical	FALSE	TRUE, FALSE	-
costs	untyped	NULL		-
subset	logical	TRUE	TRUE, FALSE	-
bands	integer	-		$[0, 1000]$
winnow	logical	FALSE	TRUE, FALSE	-
noGlobalPruning	logical	FALSE	TRUE, FALSE	-
CF	numeric	0.25		$[0, 1]$
minCases	integer	2		$[0, \infty)$
fuzzyThreshold	logical	FALSE	TRUE, FALSE	-
sample	numeric	0		$[0, 0.999]$
seed	integer	-		$(-\infty, \infty)$
earlyStopping	logical	TRUE	TRUE, FALSE	-
label	untyped	"outcome"		-
na.action	untyped	"stats::na.pass"		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifC50`

Methods

Public methods:

- `LearnerClassifC50$new()`
- `LearnerClassifC50$clone()`

`LearnerClassifC50$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifC50$new()
```

`LearnerClassifC50$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifC50$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

henrifnk

References

Quinlan, Ross J (2014). *C4. 5: programs for machine learning*. Elsevier.

---

mlr_learners_classif.catboost

Gradient Boosted Decision Trees Classification Learner

Description

Gradient boosting algorithm that also supports categorical data. Calls `catboost::catboost.train()` from package 'catboost'.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.catboost")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3extralearners**, **catboost**

Parameters

Id	Type	Default	Levels
loss_function_twoclass	character	Logloss	Logloss, CrossEntropy
loss_function_multiclass	character	MultiClass	MultiClass, MultiClassOneVsAll
learning_rate	numeric	0.03	
random_seed	integer	0	
l2_leaf_reg	numeric	3	
bootstrap_type	character	-	Bayesian, Bernoulli, MVS, Poisson, No
bagging_temperature	numeric	1	
subsample	numeric	-	
sampling_frequency	character	PerTreeLevel	PerTree, PerTreeLevel
sampling_unit	character	Object	Object, Group
mvs_reg	numeric	-	
random_strength	numeric	1	
depth	integer	6	
grow_policy	character	SymmetricTree	SymmetricTree, Depthwise, Lossguide
min_data_in_leaf	integer	1	
max_leaves	integer	31	
ignored_features	untyped	NULL	
one_hot_max_size	untyped	FALSE	
has_time	logical	FALSE	TRUE, FALSE
rsm	numeric	1	
nan_mode	character	Min	Min, Max
fold_permutation_block	integer	-	
leaf_estimation_method	character	-	Newton, Gradient, Exact
leaf_estimation_iterations	integer	-	
leaf_estimation_backtracking	character	AnyImprovement	No, AnyImprovement, Armijo
fold_len_multiplier	numeric	2	
approx_on_full_history	logical	TRUE	TRUE, FALSE
class_weights	untyped	-	
auto_class_weights	character	None	None, Balanced, SqrtBalanced
boosting_type	character	-	Ordered, Plain
boost_from_average	logical	-	TRUE, FALSE
langevin	logical	FALSE	TRUE, FALSE
diffusion_temperature	numeric	10000	
score_function	character	Cosine	Cosine, L2, NewtonCosine, NewtonL2
monotone_constraints	untyped	-	
feature_weights	untyped	-	
first_feature_use_penalties	untyped	-	
penalties_coefficient	numeric	1	

per_object_feature_penalties	untyped	-	
model_shrink_rate	numeric	-	
model_shrink_mode	character	-	Constant, Decreasing
target_border	numeric	-	
border_count	integer	-	
feature_border_type	character	GreedyLogSum	Median, Uniform, UniformAndQuantiles, MaxLogSum, Mi
per_float_feature_quantization	untyped	-	
classes_count	integer	-	
thread_count	integer	1	
task_type	character	CPU	CPU, GPU
devices	untyped	-	
logging_level	character	Silent	Silent, Verbose, Info, Debug
metric_period	integer	1	
train_dir	untyped	"catboost_info"	
model_size_reg	numeric	0.5	
allow_writing_files	logical	FALSE	TRUE, FALSE
save_snapshot	logical	FALSE	TRUE, FALSE
snapshot_file	untyped	-	
snapshot_interval	integer	600	
simple_ctr	untyped	-	
combinations_ctr	untyped	-	
ctr_target_border_count	integer	-	
counter_calc_method	character	Full	SkipTest, Full
max_ctr_complexity	integer	-	
ctr_leaf_count_limit	integer	-	
store_all_simple_ctr	logical	FALSE	TRUE, FALSE
final_ctr_computation_mode	character	Default	Default, Skip
verbose	logical	FALSE	TRUE, FALSE
ntree_start	integer	0	
ntree_end	integer	0	
early_stopping_rounds	integer	-	
eval_metric	untyped	-	
use_best_model	logical	-	TRUE, FALSE
iterations	integer	1000	

Installation

See https://catboost.ai/en/docs/concepts/r-installation.

Initial parameter values

- `logging_level`:
  - Actual default: "Verbose"
  - Adjusted default: "Silent"
  - Reason for change: consistent with other mlr3 learners

- thread_count:
  - Actual default: -1
  - Adjusted default: 1
  - Reason for change: consistent with other mlr3 learners
- allow_writing_files:
  - Actual default: TRUE
  - Adjusted default: FALSE
  - Reason for change: consistent with other mlr3 learners
- save_snapshot:
  - Actual default: TRUE
  - Adjusted default: FALSE
  - Reason for change: consistent with other mlr3 learners

Early stopping

Early stopping can be used to find the optimal number of boosting rounds. Set early_stopping_rounds to an integer value to monitor the performance of the model on the validation set while training. For information on how to configure the validation set, see the *Validation* section of mlr3::Learner.

Super classes

mlr3::Learner -> mlr3::LearnerClassif -> LearnerClassifCatboost

Active bindings

internal_valid_scores The validation scores for the eval_metric and the loss_function, evaluated on the internal validation data with all tree_count trees, i.e. the trees that are also used for prediction.

best_valid_scores The validation scores of the best iteration. Because use_best_model truncates the model to the best iteration, these are identical to $internal_valid_scores whenever early stopping is activated. If early stopping is not activated or use_best_model is FALSE, no best iteration is tracked and this is an empty list.

internal_tuned_values Returns the early stopped iterations if early_stopping_rounds was set during training.

validate How to construct the internal validation data. This parameter can be either NULL, a ratio, "test", or "predefined".

Methods

Public methods:

- LearnerClassifCatboost$new()
- LearnerClassifCatboost$importance()
- LearnerClassifCatboost$clone()

LearnerClassifCatboost$new(): Create a LearnerClassifCatboost object.

Usage:

```
LearnerClassifCatboost$new()
```

`LearnerClassifCatboost$importance()`: The importance scores are calculated using `catboost::catboost.get_feat` setting `type = "FeatureImportance"`, returned for 'all'.

Usage:

```
LearnerClassifCatboost$importance()
```

Returns: Named `numeric()`.

`LearnerClassifCatboost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifCatboost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sumny

References

Dorogush, Veronika A, Ershov, Vasily, Gulin, Andrey (2018). "CatBoost: gradient boosting with categorical features support." *arXiv preprint arXiv:1810.11363*.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.catboost", iterations = 10)
print(learner)

# Define a Task
task = tsk("sonar")
```

```
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.cforest`

Classification Conditional Random Forest Learner

Description

A random forest based on conditional inference trees (`partykit::ctree()`). Calls `partykit::cforest()` from **partykit**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.cforest")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **partykit**, **sandwich**, **coin**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
replace	logical	FALSE	TRUE, FALSE	-
fraction	numeric	0.632		$[0, 1]$
mtry	integer	-		$[0, \infty)$
mtryratio	numeric	-		$[0, 1]$
applyfun	untyped	-		-

cores	integer	NULL		$(-\infty, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
cluster	untyped	-		-
scores	untyped	-		-
teststat	character	quadratic	quadratic, maximum	-
splitstat	character	quadratic	quadratic, maximum	-
splittest	logical	FALSE	TRUE, FALSE	-
testtype	character	Univariate	Bonferroni, MonteCarlo, Univariate, Teststatistic	-
nmax	untyped	-		-
pargs	untyped	-		-
alpha	numeric	0.05		$[0, 1]$
mincriterion	numeric	0		$[0, 1]$
logmincriterion	numeric	0		$(-\infty, \infty)$
minsplit	integer	20		$[1, \infty)$
minbucket	integer	7		$[1, \infty)$
minprob	numeric	0.01		$[0, 1]$
stump	logical	FALSE	TRUE, FALSE	-
lookahead	logical	FALSE	TRUE, FALSE	-
MIA	logical	FALSE	TRUE, FALSE	-
nresample	integer	9999		$[1, \infty)$
tol	numeric	1.490116e-08		$[0, \infty)$
maxsurrogate	integer	0		$[0, \infty)$
numsurrogate	logical	FALSE	TRUE, FALSE	-
maxdepth	integer	Inf		$[0, \infty)$
multiway	logical	FALSE	TRUE, FALSE	-
splittry	integer	2		$[0, \infty)$
intersplit	logical	FALSE	TRUE, FALSE	-
majority	logical	FALSE	TRUE, FALSE	-
caseweights	logical	TRUE	TRUE, FALSE	-
saveinfo	logical	FALSE	TRUE, FALSE	-
update	logical	FALSE	TRUE, FALSE	-
splitflavour	character	ctree	ctree, exhaustive	-
maxvar	integer	-		$[1, \infty)$
OOB	logical	FALSE	TRUE, FALSE	-
simplify	logical	TRUE	TRUE, FALSE	-
scale	logical	TRUE	TRUE, FALSE	-
nperm	integer	1		$[0, \infty)$
risk	character	loglik	loglik, misclassification	-
conditional	logical	FALSE	TRUE, FALSE	-
threshold	numeric	0.2		$(-\infty, \infty)$

Custom mlr3 parameters

- `mtry`:
  - This hyperparameter can alternatively be set via the added hyperparameter `mtryratio` as

`mtry = max(ceiling(mtryratio * n_features), 1)`. Note that `mtry` and `mtryratio` are mutually exclusive.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifCForest`

Methods

Public methods:

- `LearnerClassifCForest$new()`
- `LearnerClassifCForest$oob_error()`
- `LearnerClassifCForest$clone()`

`LearnerClassifCForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifCForest$new()
```

`LearnerClassifCForest$oob_error()`: The importance scores are calculated using `partykit::varimp()`. The out-of-bag error, calculated using the OOB predictions from `partykit`.

Usage:

```
LearnerClassifCForest$oob_error()
```

Returns: `numeric(1)`.

`LearnerClassifCForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifCForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sumny

References

Hothorn T, Zeileis A (2015). "partykit: A Modular Toolkit for Recursive Partytioning in R." *Journal of Machine Learning Research*, **16**(118), 3905-3909. http://jmlr.org/papers/v16/hothorn15a.html.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. doi:10.1198/106186006x133933. https://doi.org/10.1198/106186006x133933.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.cforest", ntree = 50L)
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.ctree

Classification Conditional Inference Tree Learner

Description

Classification Partition Tree where a significance test is used to determine the univariate splits. Calls `partykit::ctree()` from **partykit**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.ctree")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3extralearners**, **partykit**, **sandwich**, **coin**

Parameters

Id	Type	Default	Levels	Range
teststat	character	quadratic	quadratic, maximum	-
splitstat	character	quadratic	quadratic, maximum	-
splittest	logical	FALSE	TRUE, FALSE	-
testtype	character	Bonferroni	Bonferroni, MonteCarlo, Univariate, Teststatistic	-
nmax	untyped	-		-
alpha	numeric	0.05		$[0, 1]$
mincriterion	numeric	0.95		$[0, 1]$
logmincriterion	numeric	-		$(-\infty, \infty)$
minsplit	integer	20		$[1, \infty)$
minbucket	integer	7		$[1, \infty)$
minprob	numeric	0.01		$[0, 1]$
stump	logical	FALSE	TRUE, FALSE	-
lookahead	logical	FALSE	TRUE, FALSE	-
MIA	logical	FALSE	TRUE, FALSE	-
nresample	integer	9999		$[1, \infty)$
tol	numeric	-		$[0, \infty)$
maxsurrogate	integer	0		$[0, \infty)$
numsurrogate	logical	FALSE	TRUE, FALSE	-
mtry	integer	Inf		$[0, \infty)$
maxdepth	integer	Inf		$[0, \infty)$
multiway	logical	FALSE	TRUE, FALSE	-
splittry	integer	2		$[0, \infty)$
intersplit	logical	FALSE	TRUE, FALSE	-
majority	logical	FALSE	TRUE, FALSE	-
caseweights	logical	FALSE	TRUE, FALSE	-
maxvar	integer	-		$[1, \infty)$
applyfun	untyped	-		-
cores	integer	NULL		$(-\infty, \infty)$
saveinfo	logical	TRUE	TRUE, FALSE	-
update	logical	FALSE	TRUE, FALSE	-
splitflavour	character	ctree	ctree, exhaustive	-
cluster	untyped	-		-

scores	untyped	-		-
doFit	logical	TRUE	TRUE, FALSE	-
maxpts	integer	25000		$(-\infty, \infty)$
abseps	numeric	0.001		$[0, \infty)$
releps	numeric	0		$[0, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifCTree`

Methods

Public methods:

- `LearnerClassifCTree$new()`
- `LearnerClassifCTree$clone()`

`LearnerClassifCTree$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifCTree$new()
```

`LearnerClassifCTree$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifCTree$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

sumny

References

Hothorn T, Zeileis A (2015). "partykit: A Modular Toolkit for Recursive Partytioning in R." *Journal of Machine Learning Research*, **16**(118), 3905-3909. http://jmlr.org/papers/v16/hothorn15a.html.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. doi:10.1198/106186006x133933. https://doi.org/10.1198/106186006x133933.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.ctree")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.cv_ncvreg`

Cross-Validated MCP- or SCAD-penalized Classification Learner

---

Description

MCP- or SCAD-penalized binary classification model with k-fold cross validation for choosing the regularization parameter lambda. Calls `ncvreg::cv.ncvreg()` from package **ncvreg** with `family` set to `"binomial"`. Covariates are internally standardized by `ncvreg` prior to model fitting.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.cv_ncvreg")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **ncvreg**

Parameters

Id	Type	Default	Levels	Range
penalty	character	MCP	MCP, SCAD, lasso	-
gamma	numeric	-		$[0, \infty)$
alpha	numeric	1		$[0, 1]$
lambda.min	numeric	-		$[0, \infty)$
nlambda	integer	100		$[1, \infty)$
lambda	untyped	-		-
eps	numeric	1e-04		$(-\infty, \infty)$
max.iter	integer	10000		$[1, \infty)$
convex	logical	TRUE	TRUE, FALSE	-
dfmax	integer	-		$[0, \infty)$
penalty.factor	untyped	-		-
warn	logical	TRUE	TRUE, FALSE	-
returnX	logical	TRUE	TRUE, FALSE	-
cluster	untyped	-		-
nfolds	integer	10		$(-\infty, \infty)$
fold	untyped	-		-
returnY	logical	FALSE	TRUE, FALSE	-
trace	logical	FALSE	TRUE, FALSE	-
pred_lambda	numeric	-		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifNCVreg`

Methods

Public methods:

- `LearnerClassifNCVreg$new()`

- `LearnerClassifNCVreg$selected_features()`
- `LearnerClassifNCVreg$clone()`

`LearnerClassifNCVreg$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifNCVreg$new()
```

`LearnerClassifNCVreg$selected_features()`: Returns the set of selected features which have non-zero coefficients. Calls the internal `coef.cv.ncvreg()` function.

Usage:

```
LearnerClassifNCVreg$selected_features(lambda = NULL)
```

Arguments:

`lambda (numeric(1))`
Custom `lambda`, defaults to the lambda with the minimum CV error.

Returns: (`character()`) vector of feature names.

`LearnerClassifNCVreg$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifNCVreg$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bblodfon

References

Breheny P, Huang J (2011). "Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection." *Annals of Applied Statistics*, **5**(1), 232–253. ISSN 1932-6157. doi:10.1214/10AOAS388.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.cv_ncvreg")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.dbnDNN

Deep Neural Network with DBN Pretraining Learner

Description

Deep neural network with weights initialized by a deep belief network. Calls `deepnet::dbn.dnn.train()` from **deepnet**.

This learner works with tasks that have at least two features.

Initial parameter values

- `output` is set to `"softmax"` to enable class probabilities.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.dbnDNN")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “integer”, “numeric”
- Required Packages: **mlr3**, **deepnet**

Parameters

Id	Type	Default	Levels	Range
hidden	untyped	-		-
activationfun	character	-	sigm, linear, tanh	-
learningrate	numeric	-		$[0, \infty)$
momentum	numeric	-		$[0, \infty)$
learningrate_scale	numeric	-		$[0, \infty)$
numepochs	integer	-		$[1, \infty)$
batchsize	integer	-		$[1, \infty)$
output	character	-	sigm, linear, softmax	-
hidden_dropout	numeric	-		$[0, 1]$
visible_dropout	numeric	-		$[0, 1]$
cd	integer	-		$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifDbnDNN`

Methods

Public methods:

- `LearnerClassifDbnDNN$new()`
- `LearnerClassifDbnDNN$clone()`

`LearnerClassifDbnDNN$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifDbnDNN$new()
```

`LearnerClassifDbnDNN$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifDbnDNN$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Rong X (2014). *deepnet: Deep Learning Toolkit in R*. R package version 0.2.1, https://CRAN.R-project.org/package=deepnet.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.dbnDNN")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.decision_stump`

Classification Decision Stump Learner

Description

Decision Stump Learner. Calls `RWeka::DecisionStump()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.decision_stump")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifDecisionStump`

Active bindings

`marshaled (logical(1))`
: Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifDecisionStump$new()`
- `LearnerClassifDecisionStump$marshal()`
- `LearnerClassifDecisionStump$unmarshal()`
- `LearnerClassifDecisionStump$clone()`

`LearnerClassifDecisionStump$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifDecisionStump$new()
```

`LearnerClassifDecisionStump$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifDecisionStump$marshal(...)
```

Arguments:

`...` (any)
: Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifDecisionStump$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifDecisionStump$unmarshal(...)
```

Arguments:

`...` (any)
: Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifDecisionStump$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifDecisionStump$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.decision_stump")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.decision_table`

Classification Decision Table Learner

Description

Simple Decision Table majority classifier. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Initial parameter values

- `E`:
  - Has only 2 out of 4 original evaluation measures : acc and auc with acc being the default
  - Reason for change: this learner should only contain evaluation measures appropriate for classification tasks

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- `P_best`:
  - original id: P
- `D_best`:
  - original id: D
- `N_best`:
  - original id: N
- `S_best`:
  - original id: S
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.decision_table")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
S	character	BestFirst	BestFirst, GreedyStepwise	-
X	integer	1		$(-\infty, \infty)$
E	character	acc	acc, auc	-
I	logical	-	TRUE, FALSE	-
R	logical	-	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
P_best	untyped	-		-
D_best	character	1	0, 1, 2	-
N_best	integer	-		$(-\infty, \infty)$
S_best	integer	1		$(-\infty, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifDecisionTable`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifDecisionTable$new()`
- `LearnerClassifDecisionTable$marshal()`
- `LearnerClassifDecisionTable$unmarshal()`
- `LearnerClassifDecisionTable$clone()`

`LearnerClassifDecisionTable$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifDecisionTable$new()
```

`LearnerClassifDecisionTable$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifDecisionTable$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifDecisionTable$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifDecisionTable$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifDecisionTable$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifDecisionTable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Kohavi R (1995). "The Power of Decision Tables." In *8th European Conference on Machine Learning*, 174–189.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.decision_table")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.dlda`
Classification Diagonal Linear Discriminant Analysis Learner

Description

Diagonal Linear Discriminant Analysis classifier. Belongs to the family of Naive Bayes classifiers, where the distributions of each class are assumed to be multivariate normal and to share a common covariance matrix. Off-diagonal elements of the pooled sample covariance matrix are set to zero Calls `sparsediscrim::lda_diag()` from **sparsediscrim**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.dlda")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **sparsediscrim**

Parameters

Id	Type	Default
prior	untyped	NULL

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifDiagLda`

Methods

Public methods:

- `LearnerClassifDiagLda$new()`
- `LearnerClassifDiagLda$clone()`

`LearnerClassifDiagLda$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifDiagLda$new()
```

`LearnerClassifDiagLda$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifDiagLda$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Dudoit S, Fridlyand J, Speed TP (2002). "Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data." *Journal of the American Statistical Association*, **97**(457), 77–87. doi:10.1198/016214502753479248.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.

- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.dlda")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.earth`

Classification MARS (Multivariate Adaptive Regression Splines) Learner

---

Description

This is an alternative implementation of MARS (Multivariate Adaptive Regression Splines). The classification problem is solved by 0-1 encoding of the two-class targets and setting the decision threshold to p = 0.5 during the prediction phase. MARS is trademarked and thus not used as the name. The name "earth" stands for "Enhanced Adaptive Regression Through Hinges".

Details

Methods for variance estimations are not yet implemented.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.earth")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3extralearners**, **earth**

Parameters

Id	Type	Default	Levels	Range
wp	untyped	NULL		-
offset	untyped	NULL		-
keepxy	logical	FALSE	TRUE, FALSE	-
trace	character	0	0, .3, .5, 1, 2, 3, 4, 5	-
degree	integer	1		$[1, \infty)$
penalty	numeric	2		$[-1, \infty)$
nk	untyped	NULL		-
thresh	numeric	0.001		$(-\infty, \infty)$
minspan	numeric	0		$[0, \infty)$
endspan	numeric	0		$[0, \infty)$
newvar.penalty	numeric	0		$[0, \infty)$
fast.k	integer	20		$[0, \infty)$
fast.beta	integer	1		$[0, 1]$
linpreds	untyped	FALSE		-
allowed	untyped	-		-
pmethod	character	backward	backward, none, exhaustive, forward, seqrep, cv	-
nprune	integer	-		$[0, \infty)$
nfold	integer	0		$[0, \infty)$
ncross	integer	1		$[0, \infty)$
stratify	logical	TRUE	TRUE, FALSE	-
varmod.method	character	none	none, const, lm, rlm, earth, gam, power, power0, x.lm, x.rlm, [...]	-
varmod.exponent	numeric	1		$(-\infty, \infty)$
varmod.conv	numeric	1		$[0, 1]$
varmod.clamp	numeric	0.1		$(-\infty, \infty)$
varmod.minspan	numeric	-3		$(-\infty, \infty)$
Scale.y	logical	FALSE	TRUE, FALSE	-
Adjust.endspan	numeric	2		$(-\infty, \infty)$
Auto.linpreds	logical	TRUE	TRUE, FALSE	-
Force.weights	logical	FALSE	TRUE, FALSE	-
Use.beta.cache	logical	TRUE	TRUE, FALSE	-
Force.xtx.prune	logical	FALSE	TRUE, FALSE	-
Get.leverages	logical	TRUE	TRUE, FALSE	-

Exhaustive.tol	numeric	1e-10	$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifEarth`

Methods

Public methods:

- `LearnerClassifEarth$new()`
- `LearnerClassifEarth$clone()`

`LearnerClassifEarth$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifEarth$new()
```

`LearnerClassifEarth$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifEarth$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

pkopper

References

Milborrow, Stephen, Hastie, T, Tibshirani, R (2014). "Earth: multivariate adaptive regression spline models." *R package version*, **3**(7).

Friedman, H J (1991). "Multivariate adaptive regression splines." *The annals of statistics*, **19**(1), 1–67.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.earth")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.evtree

Classification Decision Tree Learner

Description

Evolutionary learning of globally optimal classification trees. Calls `evtree::evtree()` from**evtree**.

Initial parameter values

`pmutatemajor`, `pmutateminor`, `pcrossover`, `psplit`, and `pprune`, are scaled internally to sum to 100.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifEvtree`

Methods

Public methods:

- `LearnerClassifEvtree$new()`
- `LearnerClassifEvtree$clone()`

`LearnerClassifEvtree$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifEvtree$new()
```

`LearnerClassifEvtree$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifEvtree$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.evtree")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.exhaustive_search`
Logistic Regression with Exhaustive Search

Description

Logistic Regression, where the features used in the model are selected by running Exhaustive Search.

Calls `ExhaustiveSearch::ExhaustiveSearch()` from **ExhaustiveSearch**.

Initial parameter values

- `family`:
  - Actual default: NULL
  - Adjusted default: "binomial"
  - Reason for change: To comply with mlr3 architecture, we differentiate between classification and regression learners.
- `nThreads`:
  - Actual default: NULL
  - Adjusted default: 1
  - Reason for change: Suppressing the automatic internal parallelization if `cv.folds` > 0.
- `quietly`:
  - Actual default: FALSE
  - Adjusted default: TRUE
  - Reason for change: Suppression of constant printing to console

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.exhaustive_search")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3extralearners**, **ExhaustiveSearch**

Parameters

Id	Type	Default	Levels	Range
family	character	-	gaussian, binomial	-
performanceMeasure	character	-	MSE, AIC	-
combsUpTo	integer	-		$[1, \infty)$
nResults	integer	5000		$[1, \infty)$
nThreads	integer	-		$[1, \infty)$
testSetIDs	integer	-		$[1, \infty)$
errorVal	untyped	-1L		-
quietly	logical	-	TRUE, FALSE	-
checkLarge	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifExhaustiveSearch`

Methods

Public methods:

- `LearnerClassifExhaustiveSearch$new()`
- `LearnerClassifExhaustiveSearch$selected_features()`
- `LearnerClassifExhaustiveSearch$clone()`

`LearnerClassifExhaustiveSearch$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifExhaustiveSearch$new()
```

`LearnerClassifExhaustiveSearch$selected_features()`: Extracts selected features of this learner.

Usage:

```
LearnerClassifExhaustiveSearch$selected_features()
```

`LearnerClassifExhaustiveSearch$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifExhaustiveSearch$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ngerman

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# define the learner
learner = lrn("classif.exhaustive_search", predict_type = "prob", combsUpTo = 3)

# define the task
# and subset to 3 features to speed up the example
tsk_sonar = tsk("sonar")$select(c("V1", "V2", "V3"))

# train the learner
learner$train(tsk_sonar)

# extract selected features
learner$selected_features()

# predict on training task
learner$predict(tsk_sonar)
```

---

mlr_learners_classif.fastai

Classification Neural Network Learner

Description

Simple and fast neural nets for tabular data classification. Calls `fastai::tabular_learner()` from package **fastai**.

Installation

The Python dependencies are automatically installed via `reticulate::py_require()`. See Installing Python Dependencies for more details. You can manually specify a virtual environment by calling `reticulate::use_virtualenv()` prior to calling the `$train()` function. In this virtual environment, the `fastai` package and its dependencies must be installed.

Saving a Learner

In order to save a `lrn("classif.fastai")` for later usage, it is necessary to call the `$marshal()` method on the `Learner` before writing it to disk, as the object will otherwise not be saved correctly. After loading a marshaled `lrn("classif.fastai")` into R again, you then need to call `$unmarshal()` to transform it into a useable state.

Initial parameter values

- `n_epoch`: Needs to be set for `fastai::fit()` to work. If no value is given, it is set to 5.
- `eval_metric`: Needs to be set for `fastai::predict()` to work. If no value is given, it is set to `fastai::accuracy()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.fastai")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **fastai**, **reticulate**

Parameters

Id	Type	Default	Levels	Range
act_cls	untyped	-		-
bn_cont	logical	TRUE	TRUE, FALSE	-
bn_final	logical	FALSE	TRUE, FALSE	-
drop_last	logical	FALSE	TRUE, FALSE	-
embed_p	numeric	0		$[0, 1]$
emb_szs	untyped	NULL		-
n_epoch	integer	5		$[1, \infty)$
eval_metric	untyped	-		-
layers	untyped	-		-
loss_func	untyped	-		-
lr	numeric	0.001		$[0, \infty)$
metrics	untyped	-		-
n_out	integer	-		$(-\infty, \infty)$
num_workers	integer	-		$(-\infty, \infty)$
opt_func	untyped	-		-
patience	integer	1		$[1, \infty)$
pin_memory	logical	TRUE	TRUE, FALSE	-
procs	untyped	NULL		-
ps	untyped	NULL		-

shuffle	logical	FALSE	TRUE, FALSE	-
train_bn	logical	TRUE	TRUE, FALSE	-
wd	integer	-		$[0, \infty)$
wd_bn_bias	logical	FALSE	TRUE, FALSE	-
use_bn	logical	TRUE	TRUE, FALSE	-
y_range	untyped	NULL		-
bs	integer	50		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifFastai`

Active bindings

`internal_valid_scores` (named `list()` or `NULL`) The validation scores extracted from `eval_protocol` which itself is set by fitting the `fastai::tab_learner`. If early stopping is activated, this contains the validation scores of the model for the optimal `n_epoch`, otherwise the `n_epoch` for the final model.

`internal_tuned_values` (named `list()` or `NULL`) If early stopping is activated, this returns a list with `n_epoch`, which is the last epoch that yielded improvement w.r.t. the `patience`, extracted by `max(eval_protocol$epoch)+1`

`validate` (`numeric(1)` or `character(1)` or `NULL`) How to construct the internal validation data. This parameter can be either `NULL`, a ratio, `"test"`, or `"predefined"`.

`marshaled` (`logical(1)`) Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifFastai$new()`
- `LearnerClassifFastai$marshal()`
- `LearnerClassifFastai$unmarshal()`
- `LearnerClassifFastai$clone()`

`LearnerClassifFastai$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifFastai$new()
```

`LearnerClassifFastai$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifFastai$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifFastai$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifFastai$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifFastai$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifFastai$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Howard, Jeremy, Gugger, Sylvain (2020). "Fastai: A Layered API for Deep Learning." *Information*, **11**(2), 108. ISSN 2078-2489. doi:10.3390/info11020108.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

`mlr_learners_classif.fnn`

Fast Nearest Neighbor Classification

Description

Fast Nearest Neighbor Classification. Calls `FNN::knn()` from **FNN**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.fnn")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **FNN**

Parameters

Id	Type	Default	Levels	Range
k	integer	1		$[1, \infty)$
algorithm	character	kd_tree	kd_tree, cover_tree, brute	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifFNN`

Methods

Public methods:

- `LearnerClassifFNN$new()`
- `LearnerClassifFNN$clone()`

`LearnerClassifFNN$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifFNN$new()
```

`LearnerClassifFNN$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifFNN$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Boltz, Sylvain, Debreuve, Eric, Barlaud, Michel (2007). "kNN-based high-dimensional Kullback-Leibler distance for tracking." In *Eighth International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS'07)*, 16–16. IEEE.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.fnn")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)
```

```
# Score the predictions
predictions$score()
```

`mlr_learners_classif.gam`

Classification Generalized Additive Model Learner

Description

Generalized additive models. Calls `mgcv::gam()` from package **mgcv** with `family` set to `binomial`.
Multilabel classification is not implemented yet.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.gam")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **mgcv**

Parameters

Id	Type	Default	Levels	Range
formula	untyped	-		-
method	character	GCV.Cp	GCV.Cp, GACV.Cp, REML, P-REML, ML, P-ML	-
optimizer	untyped	c("outer", "newton")		-
scale	numeric	0		$(-\infty, \infty)$
select	logical	FALSE	TRUE, FALSE	-
knots	untyped	NULL		-
sp	untyped	NULL		-
min.sp	untyped	NULL		-
H	untyped	NULL		-
gamma	numeric	1		$[1, \infty)$
paraPen	untyped	NULL		-
G	untyped	NULL		-
in.out	untyped	NULL		-
drop.unused.levels	logical	TRUE	TRUE, FALSE	-
drop.intercept	logical	FALSE	TRUE, FALSE	-

nei	untyped	-		-
nthreads	integer	1		$[1, \infty)$
irls.reg	numeric	0		$[0, \infty)$
epsilon	numeric	1e-07		$[0, \infty)$
maxit	integer	200		$(-\infty, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
mgcv.tol	numeric	1e-07		$[0, \infty)$
mgcv.half	integer	15		$[0, \infty)$
rank.tol	numeric	1.490116e-08		$[0, \infty)$
nlm	untyped	list()		-
optim	untyped	list()		-
newton	untyped	list()		-
idLinksBases	logical	TRUE	TRUE, FALSE	-
scalePenalty	logical	TRUE	TRUE, FALSE	-
efs.lspmax	integer	15		$[0, \infty)$
efs.tol	numeric	0.1		$[0, \infty)$
scale.est	character	fletcher	fletcher, pearson, deviance	-
edge.correct	logical	FALSE	TRUE, FALSE	-
ncv.threads	integer	1		$[1, \infty)$
block.size	integer	1000		$(-\infty, \infty)$
unconditional	logical	FALSE	TRUE, FALSE	-

Formula

A gam formula specific to the task at hand is required for the `formula` parameter (see example and `?mgcv::formula.gam`). Beware, if no formula is provided, a fallback formula is used that will make the gam behave like a glm (this behavior is required for the unit tests). Only features specified in the formula will be used, superseding columns with col_roles "feature" in the task.

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated during training via the `offset` argument in `mgcv::gam()`. No offset is applied during prediction for this learner.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifGam`

Methods

Public methods:

- `LearnerClassifGam$new()`
- `LearnerClassifGam$clone()`

`LearnerClassifGam$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifGam$new()
```

`LearnerClassifGam$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifGam$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

pierrecamilleri

References

Hastie, J T, Tibshirani, J R (2017). *Generalized additive models*. Routledge.

Wood, Simon (2012). "mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML smoothness estimation."

Examples

```
# simple example
t = tsk("spam")$filter(1:1000)
l = lrn("classif.gam")
l$param_set$set_values(
  formula = type ~ s(george, k = 3) + s(charDollar, k = 3) + s(edu)
)
l$train(t)
l$model
```

---

mlr_learners_classif.gamboost

Boosted Generalized Additive Classification Learner

Description

Fit a generalized additive classification model using a boosting algorithm. Calls `mboost::gamboost()` from **mboost**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.gamboost")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **mboost**

Parameters

Id	Type	Default	Levels	Range
baselearner	character	bbs	bbs, bols, btree	-
dfbase	integer	4		$(-\infty, \infty)$
family	character	Binomial	Binomial, AdaExp, AUC, custom	-
custom.family	untyped	-		-
link	character	logit	logit, probit	-
type	character	adaboost	glm, adaboost	-
mstop	integer	100		$(-\infty, \infty)$
nu	numeric	0.1		$(-\infty, \infty)$
risk	character	inbag	inbag, oobag, none	-
oobweights	untyped	NULL		-
trace	logical	FALSE	TRUE, FALSE	-
stopintern	untyped	FALSE		-
na.action	untyped	stats::na.omit		-

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated via the `offset` argument in `mboost`'s training function. No offset is applied during prediction for this learner.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifGAMBoost`

Methods

Public methods:

- `LearnerClassifGAMBoost$new()`
- `LearnerClassifGAMBoost$clone()`

`LearnerClassifGAMBoost$new()`: Create a `LearnerClassifGAMBoost` object.

Usage:

```
LearnerClassifGAMBoost$new()
```

`LearnerClassifGAMBoost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifGAMBoost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Bühlmann, Peter, Yu, Bin (2003). "Boosting with the L 2 loss: regression and classification." *Journal of the American Statistical Association*, **98**(462), 324–339.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.gamboost")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)
```

```
# Score the predictions
predictions$score()
```

`mlr_learners_classif.gausspr`

Classification Gaussian Process Learner

Description

Gaussian process for classification. Calls `kernlab::gausspr()` from **kernlab**. Parameters `sigma`, `degree`, `scale`, `offset` and `order` are added to make tuning `kpar` easier. If `kpar` is provided then these new parameters are ignored. If none are provided then the default "automatic" is used for `kpar`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.gausspr")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3extralearners**, **kernlab**

Parameters

Id	Type	Default	Levels	Range
scaled	untyped	TRUE		-
kernel	character	rbfdot	rbfdot, polydot, vanilladot, tanhdot, laplacedot, besseldot, anovadot, splinedot	-
sigma	numeric	-		$(-\infty,$
degree	numeric	-		$(-\infty,$
scale	numeric	-		$(-\infty,$
offset	numeric	-		$(-\infty,$
order	numeric	-		$(-\infty,$
kpar	untyped	"automatic"		-
tol	numeric	0.001		$[0, \infty)$
fit	logical	TRUE	TRUE, FALSE	-
na.action	untyped	na.omit		-
coupler	character	minpair	minpair, pkpd	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifGausspr`

Methods

Public methods:

- `LearnerClassifGausspr$new()`
- `LearnerClassifGausspr$clone()`

`LearnerClassifGausspr$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifGausspr$new()
```

`LearnerClassifGausspr$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifGausspr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Karatzoglou, Alexandros, Smola, Alex, Hornik, Kurt, Zeileis, Achim (2004). "kernlab-an S4 package for kernel methods in R." *Journal of statistical software*, **11**(9), 1–20.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.gausspr")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.gbm`

Gradient Boosting Classification Learner

Description

Gradient Boosting Classification Algorithm. Calls `gbm::gbm()` from **gbm**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.gbm")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **gbm**

Parameters

Id	Type	Default	Levels	Range
distribution	character	bernoulli	bernoulli, adaboost, huberized, multinomial	-
n.trees	integer	100		$[1, \infty)$
interaction.depth	integer	1		$[1, \infty)$
n.minobsinnode	integer	10		$[1, \infty)$
shrinkage	numeric	0.001		$[0, \infty)$
bag.fraction	numeric	0.5		$[0, 1]$
train.fraction	numeric	1		$[0, 1]$
cv.folds	integer	0		$(-\infty, \infty)$
keep.data	logical	FALSE	TRUE, FALSE	-
verbose	logical	FALSE	TRUE, FALSE	-
n.cores	integer	1		$(-\infty, \infty)$
var.monotone	untyped	-		-

Initial parameter values

- `keep.data` is initialized to `FALSE` to save memory.
- `n.cores` is initialized to 1 to avoid conflicts with parallelization through future.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifGBM`

Methods

Public methods:

- `LearnerClassifGBM$new()`
- `LearnerClassifGBM$importance()`
- `LearnerClassifGBM$clone()`

`LearnerClassifGBM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifGBM$new()
```

`LearnerClassifGBM$importance()`: The importance scores are extracted by `gbm::relative.influence()` from the model.

Usage:

```
LearnerClassifGBM$importance()
```

Returns: Named `numeric()`.

`LearnerClassifGBM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifGBM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Friedman, H J (2002). "Stochastic gradient boosting." *Computational statistics & data analysis*, **38**(4), 367–378.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.gbm")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.glmboost`
Boosted Generalized Linear Classification Learner

Description

Fit a generalized linear classification model using a boosting algorithm. Calls `mboost::glmboost()` from **mboost**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.glmboost")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **mboost**

Parameters

Id	Type	Default	Levels	Range
family	character	Binomial	Binomial, AdaExp, AUC, custom	-
custom.family	untyped	-		-
link	character	logit	logit, probit	-
type	character	adaboost	glm, adaboost	-
center	logical	TRUE	TRUE, FALSE	-
mstop	integer	100		$(-\infty, \infty)$
nu	numeric	0.1		$(-\infty, \infty)$
risk	character	inbag	inbag, oobag, none	-
oobweights	untyped	NULL		-
trace	logical	FALSE	TRUE, FALSE	-
stopintern	untyped	FALSE		-
na.action	untyped	stats::na.omit		-
contrasts.arg	untyped	-		-

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated via the `offset` argument in `mboost`'s training function. No offset is applied during prediction for this learner.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifGLMBoost`

Methods

Public methods:

- `LearnerClassifGLMBoost$new()`
- `LearnerClassifGLMBoost$clone()`

`LearnerClassifGLMBoost$new()`: Create a `LearnerClassifGLMBoost` object.

Usage:

```
LearnerClassifGLMBoost$new()
```

`LearnerClassifGLMBoost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifGLMBoost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Bühlmann, Peter, Yu, Bin (2003). "Boosting with the L 2 loss: regression and classification." *Journal of the American Statistical Association*, **98**(462), 324–339.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.glmboost")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.glmer`

Classification Linear Mixed Effect Learner

Description

Generalized linear model with random effects. Calls `lme4::glmer()` from **lme4**.

Initial parameter values

- `family` - Is set to `stats::binomial(link = "logit")`.

Formula

Although most mlr3 learners don't allow to specify the formula manually, and automatically set it by calling `task$formula()`, this learner allows to set the `formula` because it's core functionality depends it. This means that it might not always use all features that are available in the task. Be aware, that this can sometimes lead to unexpected error messages, because mlr3 checks the compatibility between the learner and the task on **all** available features.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.glmer")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **lme4**

Parameters

Id	Type	Default	Levels
formula	untyped	-	
start	untyped	NULL	
verbose	integer	0	
contrasts	untyped	NULL	
optimizer	character	-	Nelde
restart_edge	logical	FALSE	TRUE
boundary.tol	numeric	1e-05	
calc.derivs	logical	TRUE	TRUE
autoscale	untyped	NULL	
check.nobs.vs.rankZ	character	ignore	ignore
check.nobs.vs.nlev	character	stop	ignore
check.nlev.gtreq.5	character	ignore	ignore
check.nlev.gtr.1	character	stop	ignore
check.nobs.vs.nRE	character	stop	ignore
check.rankX	character	message+drop.cols	messa
check.scaleX	character	warning	warni
check.formula.LHS	character	stop	ignore
family	untyped	"stats::binomial(link = \"logit\")"	
nAGQ	integer	1	
mustart	untyped	-	
etastart	untyped	-	
check.conv.nobsmax	numeric	10000	
check.conv.nparmax	numeric	20	
check.conv.grad	untyped	"lme4::.makeCC(\"warning\", tol = 2e-3, relTol = NULL)"	
check.conv.singular	untyped	"lme4::.makeCC(action = \"message\", tol = formals(lme4::isSingular)$tol)"	
check.conv.hess	untyped	"lme4::.makeCC(action = \"warning\", tol = 1e-6)"	
optCtrl	untyped	list()	
tolPwrss	untyped	-	
compDev	logical	TRUE	TRUE
nAGQ0initStep	logical	TRUE	TRUE
check.response.not.const	untyped	"stop"	
newparams	untyped	NULL	
re.form	untyped	NULL	
random.only	logical	FALSE	TRUE
allow.new.levels	logical	FALSE	TRUE
na.action	untyped	"stats::na.pass"	

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated during training via the `offset` argument in `lme4::lmer()`. No offset is applied during prediction for this learner.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifGlmer`

Methods

Public methods:

- `LearnerClassifGlmer$new()`
- `LearnerClassifGlmer$clone()`

`LearnerClassifGlmer$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifGlmer$new()
```

`LearnerClassifGlmer$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifGlmer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sebffischer

References

Bates, M D (2010). "lme4: Mixed-effects modeling with R."

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.glmer",
  formula = credit_risk ~ (1 | credit_history) + job + property + telephone + savings)

# Define a Task
task = tsk("german_credit")
task$select(c("credit_history", "job", "property", "telephone", "savings"))

# Train the learner
learner$train(task)

print(learner$model)
```

`mlr_learners_classif.h2o.deeplearning`

Classification H2O Deep Learning Learner

Description

Classification feed-forward multilayer artificial neural network learner. Class `h2o::h2o.deeplearning()` from package **h2o**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.h2o.deeplearning")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **h2o**

Parameters

Id	Type	Default	Levels
activation	character	Rectifier	Rectifier, Tanh, TanhWithDropout, RectifierWithDropout,
adaptive_rate	logical	TRUE	TRUE, FALSE
auc_type	character	AUTO	AUTO, NONE, MACRO_OVR, WEIGHTED_OVR, MA
autoencoder	logical	FALSE	TRUE, FALSE
average_activation	numeric	0	
balance_classes	logical	FALSE	TRUE, FALSE

categorical_encoding	character	AUTO	AUTO, Enum, OneHotInternal, OneHotExplicit, Binary, E
checkpoint	untyped	NULL	
class_sampling_factors	untyped	NULL	
classification_stop	numeric	0	
diagnostics	logical	TRUE	TRUE, FALSE
distribution	character	AUTO	AUTO, bernoulli, multinomial
elastic_averaging	logical	FALSE	TRUE, FALSE
elastic_averaging_moving_rate	numeric	0.9	
elastic_averaging_regularization	numeric	0.001	
epochs	numeric	10	
epsilon	numeric	1e-08	
export_checkpoints_dir	untyped	NULL	
export_weights_and_biases	logical	FALSE	TRUE, FALSE
fast_mode	logical	TRUE	TRUE, FALSE
force_load_balance	logical	TRUE	TRUE, FALSE
gainslift_bins	integer	-1	
hidden	untyped	c(200L, 200L)	
hidden_dropout_ratios	numeric	0.5	
ignore_const_cols	logical	TRUE	TRUE, FALSE
initial_weight_distribution	character	UniformAdaptive	UniformAdaptive, Uniform, Normal
initial_weight_scale	numeric	1	
input_dropout_ratio	numeric	0	
l1	numeric	0	
l2	numeric	0	
loss	character	Automatic	Automatic, CrossEntropy, Quadratic
max_after_balance_size	numeric	5	
max_categorical_features	integer	NULL	
max_runtime_secs	numeric	0	
max_w2	numeric	3.402823e+38	
mini_batch_size	integer	1	
missing_values_handling	character	MeanImputation	MeanImputation, Skip
momentum_ramp	numeric	1e+06	
momentum_stable	numeric	0	
momentum_start	numeric	0	
nesterov_accelerated_gradient	logical	TRUE	TRUE, FALSE
overwrite_with_best_model	logical	TRUE	TRUE, FALSE
pretrained_autoencoder	untyped	NULL	
quiet_mode	logical	TRUE	TRUE, FALSE
rate	numeric	0.005	
rate_annealing	numeric	1e-06	
rate_decay	numeric	1	
replicate_training_data	logical	TRUE	TRUE, FALSE
reproducible	logical	FALSE	TRUE, FALSE
rho	numeric	0.99	
score_duty_cycle	numeric	0.1	
score_each_iteration	logical	FALSE	TRUE, FALSE
score_interval	numeric	5	
score_training_samples	integer	10000	

score_validation_samples	integer	0	
score_validation_sampling	character	Uniform	Uniform, Stratified
seed	integer	-1	
shuffle_training_data	logical	FALSE	TRUE, FALSE
single_node_mode	logical	FALSE	TRUE, FALSE
sparse	logical	FALSE	TRUE, FALSE
sparsity_beta	numeric	0	
standardize	logical	TRUE	TRUE, FALSE
stopping_metric	character	AUTO	AUTO, logloss, AUC, lift_top_group, misclassification, A
stopping_rounds	integer	5	
stopping_tolerance	numeric	0	
target_ratio_comm_to_comp	numeric	0.05	
train_samples_per_iteration	integer	-2	
verbose	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifH2ODeeplearning`

Methods

Public methods:

- `LearnerClassifH2ODeeplearning$new()`
- `LearnerClassifH2ODeeplearning$clone()`

`LearnerClassifH2ODeeplearning$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifH2ODeeplearning$new()
```

`LearnerClassifH2ODeeplearning$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifH2ODeeplearning$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Fryda T, LeDell E, Gill N, Aiello S, Fu A, Candel A, Click C, Kraljevic T, Nykodym T, Aboyoun P, Kurka M, Malohlava M, Poirier S, Wong W (2025). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.46.0.9, https://github.com/h2oai/h2o-3.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.h2o.deeplearning", epochs = 1)
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.h2o.gbm`

Classification H2O GBM Learner

Description

Gradient boosting machine learner. Class `h2o::h2o.gbm()` from package **h2o**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.h2o.gbm")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **h2o**

Parameters

Id	Type	Default	Levels
auc_type	character	AUTO	AUTO, NONE, MACRO_OVR, WEIGHTED_OVR, MACRO
auto_rebalance	logical	TRUE	TRUE, FALSE
balance_classes	logical	FALSE	TRUE, FALSE
build_tree_one_node	logical	FALSE	TRUE, FALSE
categorical_encoding	character	AUTO	AUTO, Enum, OneHotInternal, OneHotExplicit, Binary, Eigen
check_constant_response	logical	TRUE	TRUE, FALSE
checkpoint	untyped	NULL	
class_sampling_factors	untyped	NULL	
col_sample_rate	numeric	1	
col_sample_rate_change_per_level	numeric	1	
col_sample_rate_per_tree	numeric	1	
export_checkpoints_dir	untyped	NULL	
gainslift_bins	integer	-1	
histogram_type	character	AUTO	AUTO, UniformAdaptive, Random, QuantilesGlobal, RoundR
ignore_const_cols	logical	TRUE	TRUE, FALSE
in_training_checkpoints_dir	untyped	NULL	
in_training_checkpoints_tree_interval	integer	1	
interaction_constraints	untyped	NULL	
learn_rate	numeric	0.1	
learn_rate_annealing	numeric	1	
max_abs_leafnode_pred	numeric	Inf	
max_after_balance_size	numeric	5	
max_depth	integer	5	
max_runtime_secs	numeric	0	
min_rows	integer	10	
min_split_improvement	numeric	1e-05	
monotone_constraints	untyped	NULL	
nbins	integer	20	
nbins_cats	integer	1024	
nbins_top_level	integer	1024	
ntrees	integer	50	
pred_noise_bandwidth	numeric	0	

sample_rate	numeric	1	
sample_rate_per_class	untyped	NULL	
score_each_iteration	logical	FALSE	TRUE, FALSE
score_tree_interval	integer	0	
seed	integer	-1	
stopping_metric	character	AUTO	AUTO, logloss, AUC, AUCPR, lift_top_group, misclassificatio
stopping_rounds	integer	0	
stopping_tolerance	numeric	0.001	
verbose	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifH2OGBM`

Methods

Public methods:

- `LearnerClassifH2OGBM$new()`
- `LearnerClassifH2OGBM$clone()`

`LearnerClassifH2OGBM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifH2OGBM$new()
```

`LearnerClassifH2OGBM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifH2OGBM$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

awinterstetter

References

Fryda T, LeDell E, Gill N, Aiello S, Fu A, Candel A, Click C, Kraljevic T, Nykodym T, Aboyoun P, Kurka M, Malohlava M, Poirier S, Wong W (2025). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.46.0.9, https://github.com/h2oai/h2o-3.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.h2o.gbm")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.h2o.glm`

Classification H2O GLM Learner

Description

Generalized linear classification model. Calls `h2o::h2o.glm()` from package **h2o** with `family` always set to `"binomial"`.

H2O Connection

If no running H2O connection is found, the learner will automatically start a local H2O server on `127.0.0.1` via `h2o::h2o.init()`. If you want to connect to a remote H2O cluster, call `h2o::h2o.init()` with the appropriate arguments before training or predicting.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.h2o.glm")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **h2o**

Parameters

Id	Type	Default	Levels
alpha	numeric	0.5	
balance_classes	logical	FALSE	TRUE, FALSE
beta_constraints	untyped	NULL	
beta_epsilon	numeric	1e-04	
build_null_model	logical	FALSE	TRUE, FALSE
calc_like	logical	FALSE	TRUE, FALSE
checkpoint	untyped	NULL	
class_sampling_factors	untyped	NULL	
cold_start	logical	FALSE	TRUE, FALSE
compute_p_values	logical	FALSE	TRUE, FALSE
early_stopping	logical	TRUE	TRUE, FALSE
export_checkpoints_dir	untyped	NULL	
gainslift_bins	integer	-1	
generate_scoring_history	logical	FALSE	TRUE, FALSE
generate_variable_inflation_factors	logical	FALSE	TRUE, FALSE
gradient_epsilon	numeric	-1	
HGLM	logical	FALSE	TRUE, FALSE
ignore_const_cols	logical	TRUE	TRUE, FALSE
interactions	untyped	NULL	
interaction_pairs	untyped	NULL	
intercept	logical	TRUE	TRUE, FALSE
lambda	numeric	1e-05	
lambda_min_ratio	numeric	-1	
lambda_search	logical	FALSE	TRUE, FALSE
link	character	logit	family_default, logit
max_active_predictors	integer	-1	

max_after_balance_size	numeric	5	
max_iterations	integer	-1	
max_runtime_secs	numeric	0	
missing_values_handling	character	MeanImputation	MeanImputation, Skip, PlugValues
nlambdas	integer	-1	
non_negative	logical	FALSE	TRUE, FALSE
objective_epsilon	numeric	-1	
obj_reg	numeric	-1	
plug_values	untyped	NULL	
prior	numeric	-1	
random_columns	untyped	NULL	
remove_collinear_columns	logical	FALSE	TRUE, FALSE
score_each_iteration	logical	FALSE	TRUE, FALSE
score_iteration_interval	integer	-1	
seed	integer	-1	
solver	character	AUTO	AUTO, IRLSM, L_BFGS, COORDINATE_DESCENT,
standardize	logical	TRUE	TRUE, FALSE
startval	untyped	NULL	
stopping_metric	character	AUTO	AUTO, logloss, AUC, AUCPR, lift_top_group, misclass
stopping_rounds	integer	0	
stopping_tolerance	numeric	0.001	

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifH2OGLM`

Methods

Public methods:

- `LearnerClassifH2OGLM$new()`
- `LearnerClassifH2OGLM$clone()`

`LearnerClassifH2OGLM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifH2OGLM$new()
```

`LearnerClassifH2OGLM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifH2OGLM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Fryda T, LeDell E, Gill N, Aiello S, Fu A, Candel A, Click C, Kraljevic T, Nykodym T, Aboyoun P, Kurka M, Malohlava M, Poirier S, Wong W (2025). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.46.0.9, https://github.com/h2oai/h2o-3.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.h2o.glm")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.h2o.randomForest`
Classification H2O Random Forest Learner

Description

Classification random forest learner. Calls `h2o::h2o.randomForest()` from package **h2o**.

H2O Connection

If no running H2O connection is found, the learner will automatically start a local H2O server on `127.0.0.1` via `h2o::h2o.init()`. If you want to connect to a remote H2O cluster, call `h2o::h2o.init()` with the appropriate arguments before training or predicting.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.h2o.randomForest")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **h2o**

Parameters

Id	Type	Default	Levels
auc_type	character	AUTO	AUTO, NONE, MACRO_OVR, WEIGHTED_OVR, MACRO_O
balance_classes	logical	FALSE	TRUE, FALSE
binomial_double_trees	logical	FALSE	TRUE, FALSE
build_tree_one_node	logical	FALSE	TRUE, FALSE
categorical_encoding	character	AUTO	AUTO, Enum, OneHotInternal, OneHotExplicit, Binary, Eigen, L
check_constant_response	logical	TRUE	TRUE, FALSE
checkpoint	untyped	NULL	
class_sampling_factors	untyped	NULL	
col_sample_rate_change_per_level	numeric	1	
col_sample_rate_per_tree	numeric	1	
export_checkpoints_dir	untyped	NULL	
gainslift_bins	integer	-1	
histogram_type	character	AUTO	AUTO, UniformAdaptive, Random, QuantilesGlobal, RoundRobi
ignore_const_cols	logical	TRUE	TRUE, FALSE
max_after_balance_size	numeric	5	

max_depth	integer	20	
max_runtime_secs	numeric	0	
min_rows	numeric	1	
min_split_improvement	numeric	1e-05	
mtries	integer	-1	
nbins	integer	20	
nbins_cats	integer	1024	
nbins_top_level	integer	1024	
ntrees	integer	50	
sample_rate	numeric	0.632	
sample_rate_per_class	untyped	NULL	
score_each_iteration	logical	FALSE	TRUE, FALSE
score_tree_interval	integer	0	
seed	integer	-1	
stopping_metric	character	AUTO	AUTO, logloss, AUC, AUCPR, lift_top_group, misclassification,
stopping_rounds	integer	0	
stopping_tolerance	numeric	0.001	
verbose	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifH2ORandomForest`

Methods

Public methods:

- `LearnerClassifH2ORandomForest$new()`
- `LearnerClassifH2ORandomForest$clone()`

`LearnerClassifH2ORandomForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifH2ORandomForest$new()
```

`LearnerClassifH2ORandomForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifH2ORandomForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Fryda T, LeDell E, Gill N, Aiello S, Fu A, Candel A, Click C, Kraljevic T, Nykodym T, Aboyoun P, Kurka M, Malohlava M, Poirier S, Wong W (2025). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.46.0.9, https://github.com/h2oai/h2o-3.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.h2o.randomForest")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.IBk`
Classification IBk Learner

Description

Instance based algorithm: K-nearest neighbors regression. Calls `RWeka::IBk()` from **RWeka**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.IBk")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **RWeka**

Parameters

Id	Type	Default	Levels
subset	untyped	-	
na.action	untyped	-	
weight	character	-	I, F
K	integer	1	
E	logical	FALSE	TRUE, FALSE
W	integer	0	
X	logical	FALSE	TRUE, FALSE
A	character	LinearNNSearch	BallTree, CoverTree, FilteredNeighbourSearch, KDTree, Linearl
output_debug_info	logical	FALSE	TRUE, FALSE
do_not_check_capabilities	logical	FALSE	TRUE, FALSE
num_decimal_places	integer	2	
batch_size	integer	100	
options	untyped	NULL	

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern
- `weight`:
  - original id: I and F
- Reason for change: original `I` and `F` params are interdependent (`I` can only be `TRUE` when `F` is `FALSE` and vice versa). The easiest way to encode this is to combine `I` and `F` into one factor param.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifIBk`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifIBk$new()`
- `LearnerClassifIBk$marshal()`
- `LearnerClassifIBk$unmarshal()`
- `LearnerClassifIBk$clone()`

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

Usage:

```
LearnerClassifIBk$new()
```

`LearnerClassifIBk$marshal()`**:** Marshal the learner's model.

Usage:

```
LearnerClassifIBk$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifIBk$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifIBk$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifIBk$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifIBk$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

henrifnk

References

Aha, W D, Kibler, Dennis, Albert, K M (1991). "Instance-based learning algorithms." *Machine learning*, **6**(1), 37–66.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.IBk")
print(learner)

# Define a Task
task = tsk("sonar")
```

```
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.imbalanced_rfsrc`
Classification Imbalanced Random Forest Src Learner

Description

Imbalanced Random forest for classification between two classes. Calls `randomForestSRC::imbalanced.rfsrc()` from from **randomForestSRC**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.imbalanced_rfsrc")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **randomForestSRC**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
method	character	rfq	rfq, brf, standard	-
block.size	integer	10		$[1, \infty)$
fast	logical	FALSE	TRUE, FALSE	-
ratio	numeric	-		$[0, 1]$

mtry	integer	-		$[1, \infty)$
mtry.ratio	numeric	-		$[0, 1]$
nodesize	integer	15		$[1, \infty)$
nodedepth	integer	-		$[1, \infty)$
splitrule	character	gini	gini, auc, entropy	-
nsplit	integer	10		$[0, \infty)$
importance	character	FALSE	FALSE, TRUE, none, permute, random, anti	-
bootstrap	character	by.root	by.root, by.node, none, by.user	-
samptype	character	swor	swor, swr	-
samp	untyped	-		-
membership	logical	FALSE	TRUE, FALSE	-
sampsize	untyped	-		-
sampsize.ratio	numeric	-		$[0, 1]$
na.action	character	na.omit	na.omit, na.impute	-
nimpute	integer	1		$[1, \infty)$
ntime	integer	-		$[1, \infty)$
proximity	character	FALSE	FALSE, TRUE, inbag, oob, all	-
distance	character	FALSE	FALSE, TRUE, inbag, oob, all	-
forest.wt	character	FALSE	FALSE, TRUE, inbag, oob, all	-
xvar.wt	untyped	-		-
split.wt	untyped	-		-
forest	logical	TRUE	TRUE, FALSE	-
var.used	character	FALSE	FALSE, all.trees	-
split.depth	character	FALSE	FALSE, all.trees	-
seed	integer	-		$(-\infty, -1]$
do.trace	logical	FALSE	TRUE, FALSE	-
get.tree	untyped	-		-
outcome	character	train	train, test	-
ptn.count	integer	0		$[0, \infty)$
cores	integer	1		$[1, \infty)$
save.memory	logical	FALSE	TRUE, FALSE	-
perf.type	character	-	gmean, misclass, brier, none	-
case.depth	logical	FALSE	TRUE, FALSE	-
marginal.xvar	untyped	NULL		-

Custom mlr3 parameters

- `mtry`: This hyperparameter can alternatively be set via the added hyperparameter `mtry.ratio` as `mtry = max(ceiling(mtry.ratio * n_features), 1)`. Note that `mtry` and `mtry.ratio` are mutually exclusive.
- `sampsize`: This hyperparameter can alternatively be set via the added hyperparameter `sampsize.ratio` as `sampsize = max(ceiling(sampsize.ratio * n_obs), 1)`. Note that `sampsize` and `sampsize.ratio` are mutually exclusive.
- `cores`: This value is set as the option `rf.cores` during training and is set to 1 by default.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifImbalancedRandomForestSRC`

Methods

Public methods:

- `LearnerClassifImbalancedRandomForestSRC$new()`
- `LearnerClassifImbalancedRandomForestSRC$importance()`
- `LearnerClassifImbalancedRandomForestSRC$selected_features()`
- `LearnerClassifImbalancedRandomForestSRC$oob_error()`
- `LearnerClassifImbalancedRandomForestSRC$clone()`

`LearnerClassifImbalancedRandomForestSRC$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifImbalancedRandomForestSRC$new()
```

`LearnerClassifImbalancedRandomForestSRC$importance()`: The importance scores are extracted from the slot `importance`.

Usage:

```
LearnerClassifImbalancedRandomForestSRC$importance()
```

Returns: Named `numeric()`.

`LearnerClassifImbalancedRandomForestSRC$selected_features()`: Selected features are extracted from the model slot `var.used`.

Note: Due to a known issue in `randomForestSRC`, enabling `var.used = "all.trees"` causes prediction to fail. Therefore, this setting should be used exclusively for feature selection purposes and not when prediction is required.

Usage:

```
LearnerClassifImbalancedRandomForestSRC$selected_features()
```

Returns: `character()`.

`LearnerClassifImbalancedRandomForestSRC$oob_error()`: OOB error extracted from the model slot `err.rate`.

Usage:

```
LearnerClassifImbalancedRandomForestSRC$oob_error()
```

Returns: `numeric()`.

`LearnerClassifImbalancedRandomForestSRC$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifImbalancedRandomForestSRC$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

HarutyunyanLiana

References

O'Brien R, Ishwaran H (2019). "A random forests quantile classifier for class imbalanced data." *Pattern Recognition*, **90**, 232–249. doi:10.1016/j.patcog.2019.01.036.

Chao C, Leo B (2004). "Using Random Forest to Learn Imbalanced Data." *University of California, Berkeley*.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.imbalanced_rfsrc", importance = "TRUE")
print(learner)

# Define a Task
task = tsk("sonar")
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.J48`
Classification J48 Learner

Description

Decision tree algorithm. Calls `RWeka::IBk()` from **RWeka**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.J48")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
U	logical	FALSE	TRUE, FALSE	-
O	logical	FALSE	TRUE, FALSE	-
C	numeric	0.25		$[2.22044604925031e-16, 1]$
M	integer	2		$[1, \infty)$
R	logical	FALSE	TRUE, FALSE	-
N	integer	3		$[2, \infty)$
B	logical	FALSE	TRUE, FALSE	-
S	logical	FALSE	TRUE, FALSE	-
L	logical	FALSE	TRUE, FALSE	-
A	logical	FALSE	TRUE, FALSE	-
J	logical	FALSE	TRUE, FALSE	-
Q	integer	1		$[1, \infty)$
doNotMakeSplitPointActualValue	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifJ48`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifJ48$new()`
- `LearnerClassifJ48$marshal()`
- `LearnerClassifJ48$unmarshal()`
- `LearnerClassifJ48$clone()`

`LearnerClassifJ48$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifJ48$new()
```

`LearnerClassifJ48$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifJ48$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifJ48$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifJ48$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifJ48$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifJ48$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

henrifnk

References

Quinlan, Ross J (2014). *C4. 5: programs for machine learning*. Elsevier.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.J48")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.JRip`
Classification JRip Learner

Description

Repeated Incremental Pruning to Produce Error Reduction. Calls `RWeka::JRip()` from **RWeka**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.JRip")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
F	integer	3		$[2, \infty)$
N	numeric	2		$[0, \infty)$
O	integer	2		$[1, \infty)$
D	logical	FALSE	TRUE, FALSE	-
S	integer	1		$[1, \infty)$
E	logical	FALSE	TRUE, FALSE	-
P	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Parameter changes

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifJRip`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifJRip$new()`
- `LearnerClassifJRip$marshal()`
- `LearnerClassifJRip$unmarshal()`
- `LearnerClassifJRip$clone()`

`LearnerClassifJRip$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifJRip$new()
```

`LearnerClassifJRip$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifJRip$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifJRip$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifJRip$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifJRip$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifJRip$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

henrifnk

References

Cohen, W W (1995). "Fast effective rule induction." In *Machine learning proceedings 1995*, 115–123. Elsevier.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.JRip")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)
```

```
# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.knn`

Classification K-nearest Neighbor Learner

Description

K-nearest neighbor classification. Calls `class::knn()` from **class**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.knn")
```

Meta Information

- Task type: "classif"
- Predict Types: "response"
- Feature Types: "numeric"
- Required Packages: **mlr3**, **class**

Parameters

Id	Type	Default	Levels	Range
k	integer	1		$[1, \infty)$
l	numeric	0		$[0, \infty)$
prob	logical	FALSE	TRUE, FALSE	-
use.all	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifKnn`

Methods

Public methods:

- `LearnerClassifKnn$new()`
- `LearnerClassifKnn$clone()`

`LearnerClassifKnn$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifKnn$new()
```

`LearnerClassifKnn$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifKnn$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*, Fourth edition. Springer, New York. ISBN 0-387-95457-0, https://www.stats.ox.ac.uk/pub/MASS4/.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.knn")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.kstar`

Classification KStar Learner

Description

Instance-based classifier which differs from other instance-based learners in that it uses an entropy-based distance function. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.kstar")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
B	integer	20		$(-\infty, \infty)$
E	logical	-	TRUE, FALSE	-
M	character	a	a, d, m, n	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifKStar`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifKStar$new()`
- `LearnerClassifKStar$marshal()`
- `LearnerClassifKStar$unmarshal()`
- `LearnerClassifKStar$clone()`

`LearnerClassifKStar$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifKStar$new()
```

`LearnerClassifKStar$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifKStar$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifKStar$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifKStar$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifKStar$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifKStar$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Cleary JG, Trigg LE (1995). "K*: An Instance-based Learner Using an Entropic Distance Measure." In *12th International Conference on Machine Learning*, 108-114.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.kstar")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.ksvm`

Classification Kernlab Support Vector Machine

Description

Support vector machine for classification. Calls `kernlab::ksvm()` from **kernlab**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.ksvm")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **kernlab**

Parameters

Id	Type	Default	Levels	Range
scaled	logical	TRUE	TRUE, FALSE	-
type	character	C-svc	C-svc, nu-svc, C-bsvc, spoc-svc, kbb-svc	-
kernel	character	rbfdot	rbfdot, polydot, vanilladot, laplacedot, besseldot, anovadot	-
C	numeric	1		$(-\infty, \infty)$
nu	numeric	0.2		$[0, \infty)$
cache	integer	40		$[1, \infty)$
tol	numeric	0.001		$[0, \infty)$
shrinking	logical	TRUE	TRUE, FALSE	-
sigma	numeric	-		$[0, \infty)$
degree	integer	-		$[1, \infty)$
scale	numeric	-		$[0, \infty)$
order	integer	-		$(-\infty, \infty)$
offset	numeric	-		$(-\infty, \infty)$
coupler	character	minpair	minpair, pkpd	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifKSVM`

Methods

Public methods:

- `LearnerClassifKSVM$new()`
- `LearnerClassifKSVM$clone()`

`LearnerClassifKSVM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifKSVM$new()
```

`LearnerClassifKSVM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifKSVM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

mboecker

References

Karatzoglou, Alexandros, Smola, Alex, Hornik, Kurt, Zeileis, Achim (2004). "kernlab-an S4 package for kernel methods in R." *Journal of statistical software*, **11**(9), 1–20.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.ksvm")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.liblinear

LiblineaR Classification Learner

Description

L2 regularized support vector classification. Calls `LiblineaR::LiblineaR()` from **LiblineaR**.

Details

Type of SVC depends on `type` argument:

- `0` – L2-regularized logistic regression (primal)
- `1` - L2-regularized L2-loss support vector classification (dual)
- `3` - L2-regularized L1-loss support vector classification (dual)
- `2` – L2-regularized L2-loss support vector classification (primal)
- `4` – Support vector classification by Crammer and Singer
- `5` - L1-regularized L2-loss support vector classification
- `6` - L1-regularized logistic regression
- `7` - L2-regularized logistic regression (dual)

If number of records > number of features, `type = 2` is faster than `type = 1` (Hsu et al. 2003).

Note that probabilistic predictions are only available for types `0`, `6`, and `7`. The default `epsilon` value depends on the `type` parameter, see `LiblineaR::LiblineaR()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.liblinear")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “numeric”
- Required Packages: **mlr3**, **mlr3extralearners**, **LiblineaR**

Parameters

Id	Type	Default	Levels	Range
type	integer	0		$[0, 7]$
cost	numeric	1		$[0, \infty)$
epsilon	numeric	-		$[0, \infty)$
bias	numeric	1		$(-\infty, \infty)$
cross	integer	0		$[0, \infty)$
verbose	logical	FALSE	TRUE, FALSE	-
wi	untyped	NULL		-
findC	logical	FALSE	TRUE, FALSE	-
useInitC	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifLiblineaR`

Methods

Public methods:

- `LearnerClassifLiblineaR$new()`
- `LearnerClassifLiblineaR$clone()`

`LearnerClassifLiblineaR$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifLiblineaR$new()
```

`LearnerClassifLiblineaR$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifLiblineaR$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Fan, Rong-En, Chang, Kai-Wei, Hsieh, Cho-Jui, Wang, Xiang-Rui, Lin, Chih-Jen (2008). "LIBLINEAR: A library for large linear classification." *the Journal of machine Learning research*, **9**, 1871–1874.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.liblinear")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.lightgbm`
Classification LightGBM Learner

Description

Gradient boosting algorithm. Calls `lightgbm::lightgbm()` from **lightgbm**. The list of parameters can be found here and in the documentation of `lightgbm::lgb.train()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.lightgbm")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **lightgbm**

Parameters

Id	Type	Default	Levels	Range
objective	character	-	binary, multiclass, multiclassova	-
eval	untyped	-		-
verbose	integer	1		$(-\infty, \infty)$
record	logical	TRUE	TRUE, FALSE	-
eval_freq	integer	1		$[1, \infty)$
callbacks	untyped	-		-
reset_data	logical	FALSE	TRUE, FALSE	-
boosting	character	gbdt	gbdt, rf, dart, goss	-
linear_tree	logical	FALSE	TRUE, FALSE	-
learning_rate	numeric	0.1		$[0, \infty)$
num_leaves	integer	31		$[1, 131072]$
tree_learner	character	serial	serial, feature, data, voting	-
num_threads	integer	0		$[0, \infty)$
device_type	character	cpu	cpu, gpu	-
seed	integer	-		$(-\infty, \infty)$
deterministic	logical	FALSE	TRUE, FALSE	-
data_sample_strategy	character	bagging	bagging, goss	-
force_col_wise	logical	FALSE	TRUE, FALSE	-
force_row_wise	logical	FALSE	TRUE, FALSE	-
histogram_pool_size	numeric	-1		$(-\infty, \infty)$
max_depth	integer	-1		$(-\infty, \infty)$
min_data_in_leaf	integer	20		$[0, \infty)$
min_sum_hessian_in_leaf	numeric	0.001		$[0, \infty)$
bagging_fraction	numeric	1		$[0, 1]$
pos_bagging_fraction	numeric	1		$[0, 1]$
neg_bagging_fraction	numeric	1		$[0, 1]$
bagging_freq	integer	0		$[0, \infty)$
bagging_seed	integer	3		$(-\infty, \infty)$
bagging_by_query	logical	FALSE	TRUE, FALSE	-
feature_fraction	numeric	1		$[0, 1]$
feature_fraction_bynode	numeric	1		$[0, 1]$
feature_fraction_seed	integer	2		$(-\infty, \infty)$
extra_trees	logical	FALSE	TRUE, FALSE	-
extra_seed	integer	6		$(-\infty, \infty)$
max_delta_step	numeric	0		$(-\infty, \infty)$
lambda_l1	numeric	0		$[0, \infty)$
lambda_l2	numeric	0		$[0, \infty)$
linear_lambda	numeric	0		$[0, \infty)$
min_gain_to_split	numeric	0		$[0, \infty)$
drop_rate	numeric	0.1		$[0, 1]$
max_drop	integer	50		$(-\infty, \infty)$
skip_drop	numeric	0.5		$[0, 1]$
xgboost_dart_mode	logical	FALSE	TRUE, FALSE	-
uniform_drop	logical	FALSE	TRUE, FALSE	-
drop_seed	integer	4		$(-\infty, \infty)$

top_rate	numeric	0.2		$[0, 1]$
other_rate	numeric	0.1		$[0, 1]$
min_data_per_group	integer	100		$[1, \infty)$
max_cat_threshold	integer	32		$[1, \infty)$
cat_l2	numeric	10		$[0, \infty)$
cat_smooth	numeric	10		$[0, \infty)$
max_cat_to_onehot	integer	4		$[1, \infty)$
top_k	integer	20		$[1, \infty)$
monotone_constraints	untyped	NULL		-
monotone_constraints_method	character	basic	basic, intermediate, advanced	-
monotone_penalty	numeric	0		$[0, \infty)$
feature_contri	untyped	NULL		-
forcedsplits_filename	untyped	""		-
refit_decay_rate	numeric	0.9		$[0, 1]$
cegb_tradeoff	numeric	1		$[0, \infty)$
cegb_penalty_split	numeric	0		$[0, \infty)$
cegb_penalty_feature_lazy	untyped	-		-
cegb_penalty_feature_coupled	untyped	-		-
path_smooth	numeric	0		$[0, \infty)$
interaction_constraints	untyped	-		-
use_quantized_grad	logical	TRUE	TRUE, FALSE	-
num_grad_quant_bins	integer	4		$(-\infty, \infty)$
quant_train_renew_leaf	logical	FALSE	TRUE, FALSE	-
stochastic_rounding	logical	TRUE	TRUE, FALSE	-
serializable	logical	TRUE	TRUE, FALSE	-
max_bin	integer	255		$[2, \infty)$
max_bin_by_feature	untyped	NULL		-
min_data_in_bin	integer	3		$[1, \infty)$
bin_construct_sample_cnt	integer	200000		$[1, \infty)$
data_random_seed	integer	1		$(-\infty, \infty)$
is_enable_sparse	logical	TRUE	TRUE, FALSE	-
enable_bundle	logical	TRUE	TRUE, FALSE	-
use_missing	logical	TRUE	TRUE, FALSE	-
zero_as_missing	logical	FALSE	TRUE, FALSE	-
feature_pre_filter	logical	TRUE	TRUE, FALSE	-
pre_partition	logical	FALSE	TRUE, FALSE	-
two_round	logical	FALSE	TRUE, FALSE	-
forcedbins_filename	untyped	""		-
is_unbalance	logical	FALSE	TRUE, FALSE	-
scale_pos_weight	numeric	1		$[0, \infty)$
sigmoid	numeric	1		$[0, \infty)$
boost_from_average	logical	TRUE	TRUE, FALSE	-
eval_at	untyped	1:5		-
multi_error_top_k	integer	1		$[1, \infty)$
auc_mu_weights	untyped	NULL		-
num_machines	integer	1		$[1, \infty)$
local_listen_port	integer	12400		$[1, \infty)$
time_out	integer	120		$[1, \infty)$

machines	untyped	""		-
gpu_platform_id	integer	-1		$(-\infty, \infty)$
gpu_device_id	integer	-1		$(-\infty, \infty)$
gpu_device_id_list	untyped	NULL		-
gpu_use_dp	logical	FALSE	TRUE, FALSE	-
num_gpu	integer	1		$[1, \infty)$
start_iteration_predict	integer	0		$(-\infty, \infty)$
num_iteration_predict	integer	-1		$(-\infty, \infty)$
pred_early_stop	logical	FALSE	TRUE, FALSE	-
pred_early_stop_freq	integer	10		$(-\infty, \infty)$
pred_early_stop_margin	numeric	10		$(-\infty, \infty)$
num_iterations	integer	100		$[1, \infty)$
early_stopping_rounds	integer	-		$[1, \infty)$
early_stopping_min_delta	numeric	-		$[0, \infty)$
first_metric_only	logical	FALSE	TRUE, FALSE	-

Initial parameter values

- `num_threads`:
  - Actual default: 0L
  - Initial value: 1L
  - Reason for change: Prevents accidental conflicts with `future`.
- `verbose`:
  - Actual default: 1L
  - Initial value: -1L
  - Reason for change: Prevents accidental conflicts with mlr messaging system.
- `objective`:
  - Depends on the task: if binary classification, then this parameter is set to `"binary"`, otherwise `"multiclass"` and cannot be changed.

Custom mlr3 parameters

- `num_class`: This parameter is automatically inferred for multiclass tasks and does not have to be set.

Early Stopping and Validation

Early stopping can be used to find the optimal number of boosting rounds. Set `early_stopping_rounds` to an integer value to monitor the performance of the model on the validation set while training. For information on how to configure the validation set, see the *Validation* section of `mlr3::Learner`. The internal validation measure can be set the `eval` parameter which should be a list of mlr3::Measures, functions, or strings for the internal lightgbm measures. If `first_metric_only` = `FALSE` (default), the learner stops when any metric fails to improve.

Super classes

mlr3::Learner -> mlr3::LearnerClassif -> LearnerClassifLightGBM

Active bindings

internal_valid_scores The validation scores for all metrics at the best iteration (model$best_iter), which is also the iteration that LightGBM predicts with. Extracted from model$record_evals.

best_valid_scores The validation scores for all metrics at the best iteration (model$best_iter), i.e. the iteration that is also reported via $internal_tuned_values. Because LightGBM also predicts with the best iteration, these are identical to $internal_valid_scores whenever early stopping is activated. If early stopping is not activated, no best iteration is tracked and this is an empty list.

internal_tuned_values Returns the early stopped iterations if early_stopping_rounds was set during training.

validate How to construct the internal validation data. This parameter can be either NULL, a ratio, "test", or "predefined".

Methods

Public methods:

- LearnerClassifLightGBM$new()
- LearnerClassifLightGBM$importance()
- LearnerClassifLightGBM$clone()

LearnerClassifLightGBM$new(): Creates a new instance of this R6 class.

Usage:

```
LearnerClassifLightGBM$new()
```

LearnerClassifLightGBM$importance(): The importance scores are extracted from lbg.importance.

Usage:

```
LearnerClassifLightGBM$importance()
```

Returns: Named numeric().

LearnerClassifLightGBM$clone(): The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifLightGBM$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

kapsner

References

Ke, Guolin, Meng, Qi, Finley, Thomas, Wang, Taifeng, Chen, Wei, Ma, Weidong, Ye, Qiwei, Liu, Tie-Yan (2017). "Lightgbm: A highly efficient gradient boosting decision tree." *Advances in neural information processing systems*, **30**.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.lightgbm")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.LMT`

Classification Logistic Model Trees Learner

Description

Classification tree with logistic regression models at the leaves. Calls `RWeka::LMT()` from **RWeka**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.LMT")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
B	logical	FALSE	TRUE, FALSE	-
R	logical	FALSE	TRUE, FALSE	-
C	logical	FALSE	TRUE, FALSE	-
P	logical	FALSE	TRUE, FALSE	-
I	integer	-		$[1, \infty)$
M	integer	15		$[1, \infty)$
W	numeric	0		$[0, 1]$
A	logical	FALSE	TRUE, FALSE	-
doNotMakeSplitPointActualValue	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifLMT`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifLMT$new()`
- `LearnerClassifLMT$marshal()`
- `LearnerClassifLMT$unmarshal()`
- `LearnerClassifLMT$clone()`

`LearnerClassifLMT$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifLMT$new()
```

`LearnerClassifLMT$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifLMT$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifLMT$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifLMT$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifLMT$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifLMT$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

henrifnk

References

Landwehr, Niels, Hall, Mark, Frank, Eibe (2005). "Logistic model trees." *Machine learning*, **59**(1), 161–205.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.LMT")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.logistic`
Classification Logistic Regression Learner

Description

Multinomial Logistic Regression model with a ridge estimator. Calls `RWeka::Logistic()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.logistic")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
C	logical	FALSE	TRUE, FALSE	-
R	numeric	-		$(-\infty, \infty)$
M	integer	-1		$(-\infty, \infty)$
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifLogistic`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifLogistic$new()`
- `LearnerClassifLogistic$marshal()`
- `LearnerClassifLogistic$unmarshal()`
- `LearnerClassifLogistic$clone()`

`LearnerClassifLogistic$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifLogistic$new()
```

`LearnerClassifLogistic$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifLogistic$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifLogistic$unmarshal()`: Unmarshal the learner's model.

Usage:

`LearnerClassifLogistic$unmarshal(...)`

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifLogistic$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifLogistic$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

le Cessie, S., van Houwelingen, J.C. (1992). "Ridge Estimators in Logistic Regression." *Applied Statistics*, **41**(1), 191-201.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.logistic")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.lssvm` *Classification Least Squares Support Vector Machine Learner*

Description

Least squares support vector machine for classification. Calls `kernlab::lssvm()` from **kernlab**. Parameters `sigma`, `degree`, `scale`, `offset`, `order`, `length`, `lambda`, and `normalized` are added to make tuning `kpar` easier. If `kpar` is provided then these new parameters are ignored. If none are provided then the default "automatic" is used for `kpar`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.lssvm")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”
- Feature Types: “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3extralearners**, **kernlab**

Parameters

Id	Type	Default	Levels
scaled	untyped	TRUE	
kernel	character	rbfdot	rbfdot, polydot, vanilladot, tanhdot, laplacedot, besseldot, anovadot, splinedot, string
sigma	numeric	-	
degree	numeric	-	
scale	numeric	-	
offset	numeric	-	
order	numeric	-	
length	integer	-	

lambda	numeric	-	
normalized	logical	-	TRUE, FALSE
kpar	untyped	"automatic"	
tau	numeric	0.01	
reduced	logical	TRUE	TRUE, FALSE
rank	integer	-	
delta	integer	40	
tol	numeric	1e-04	
fit	logical	TRUE	TRUE, FALSE
na.action	untyped	na.omit	
coupler	character	minpair	minpair, pkpd

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifLSSVM`

Methods

Public methods:

- `LearnerClassifLSSVM$new()`
- `LearnerClassifLSSVM$clone()`

`LearnerClassifLSSVM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifLSSVM$new()
```

`LearnerClassifLSSVM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifLSSVM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Karatzoglou, Alexandros, Smola, Alex, Hornik, Kurt, Zeileis, Achim (2004). "kernlab-an S4 package for kernel methods in R." *Journal of statistical software*, **11**(9), 1–20.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.lssvm")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.lvq1

Classification Learning Vector Quantization 1

Description

Learning Vector Quantization 1. Calls `class::lvqinit()`, `class::lvq1()`, and `class::lvqtest()` from **class**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.lvq1")
```

Meta Information

- Task type: "classif"
- Predict Types: "response"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **class**

Parameters

Id	Type	Default	Range
alpha	numeric	0.03	$[0, \infty)$
k	integer	5	$[1, \infty)$
niter	integer	NULL	$[1, \infty)$
prior	untyped	NULL	-
size	integer	NULL	$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifLvq1`

Methods

Public methods:

- `LearnerClassifLvq1$new()`
- `LearnerClassifLvq1$clone()`

`LearnerClassifLvq1$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifLvq1$new()
```

`LearnerClassifLvq1$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifLvq1$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Note

The learner does not work on tasks with two target groups and one feature.

Author(s)

awinterstetter

References

Kohonen, Teuvo (1990). "The self-organizing map." *Proceedings of the IEEE*, **78**(9), 1464–1480. doi:10.1109/5.58325.

Kohonen, Teuvo (1995). *Self-Organizing Maps*, volume 30 series Springer Series in Information Sciences. Springer, Berlin. ISBN 978-3540967120.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.lvq1")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)
```

```
# Score the predictions
predictions$score()
```

`mlr_learners_classif.mda`
Classification Discriminant Analysis Learner

Description

Mixture Discriminant Analysis. Calls `mda::mda()` from **mda**.

Initial parameter values

- `keep.fitted`: Set to `FALSE` by default for speed.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.mda")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mda**

Parameters

Id	Type	Default	Levels	Range
criterion	character	misclassification	misclassification, deviance	-
dimension	integer	-		$[1, \infty)$
eps	numeric	2.220446e-16		$[0, \infty)$
iter	integer	5		$[1, \infty)$
keep.fitted	logical	TRUE	TRUE, FALSE	-
method	character	polyreg	polyreg, mars, bruto, gen.ridge	-
prior	numeric	-		$[0, 1]$
start.method	character	kmeans	kmeans, lvq	-
sub.df	integer	-		$[1, \infty)$
subclasses	integer	2		$(-\infty, \infty)$
tot.df	integer	-		$[1, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
tries	integer	5		$[1, \infty)$
weights	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifMda`

Methods

Public methods:

- `LearnerClassifMda$new()`
- `LearnerClassifMda$clone()`

`LearnerClassifMda$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifMda$new()
```

`LearnerClassifMda$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifMda$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.mda")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.mdeb

Classification Linear Discriminant Analysis Learner

Description

Minimum Distance Empirical Bayesian classification, designed for small-sample, high-dimensional data. Empirical Bayes estimator where the eigenvalues of the pooled sample covariance matrix are shrunken towards the identity matrix. Calls `sparsediscrim::lda_emp_bayes()` from FIXME: **sparsediscrim**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.mdeb")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **sparsediscrim**

Parameters

Id	Type	Default
prior	untyped	NULL

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifMdeb`

Methods

Public methods:

- `LearnerClassifMdeb$new()`
- `LearnerClassifMdeb$clone()`

`LearnerClassifMdeb$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifMdeb$new()
```

`LearnerClassifMdeb$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifMdeb$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Srivastava, M., Kubokawa, T. (2007). "Comparison of Discrimination Methods for High Dimensional Data." *Journal of the Japanese Statistical Association*, **37**(1), 123–134.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.mdeb")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.mlp_rsnns`
Classification Multilayer Perceptron Learner

Description

Calls `RSNNS::mlp()` from **RSNNS**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.mlp_rsnns")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **RSNNS**

Parameters

Id	Type	Default	Levels	Range
hiddenActFunc	untyped	"Act_Logistic"		-
initFunc	untyped	"Randomize_Weights"		-
initFuncParams	untyped	-		-
inputsTest	untyped	-		-
learnFunc	untyped	"Std_Backpropagation"		-
learnFuncParams	untyped	-		-
linOut	logical	FALSE	TRUE, FALSE	-
maxit	integer	100		$[1, \infty)$
pruneFunc	untyped	-		-
pruneFuncParams	untyped	-		-
shufflePatterns	logical	TRUE	TRUE, FALSE	-
size	untyped	5L		-
targetsTest	untyped	-		-
updateFunc	untyped	"Topological_Order"		-
updateFuncParams	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifMLPRsnns`

Methods

Public methods:

- `LearnerClassifMLPRsnns$new()`
- `LearnerClassifMLPRsnns$clone()`

`LearnerClassifMLPRsnns$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifMLPRsnns$new()
```

`LearnerClassifMLPRsnns$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifMLPRsnns$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Bergmeir, Christoph, Benitez, M. J (2023). "RSNNS: Neural Networks using the Stuttgart Neural Network Simulator (SNNS)." *R package version 0.4-17*. doi:10.32614/CRAN.package.RSNNS. https://CRAN.R-project.org/package=RSNNS.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.mlp_rsnns")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.mob`

Classification Model-based Recursive Partitioning Learner

Description

Model-based recursive partitioning algorithm. Calls `partykit::mob()` from **mob**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.mob")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3extralearners**, **partykit**, **sandwich**, **coin**

Parameters

Id	Type	Default	Levels	Range
rhs	untyped	-		-
fit	untyped	-		-
offset	untyped	-		-
cluster	untyped	-		-
alpha	numeric	0.05		$[0, 1]$
bonferroni	logical	TRUE	TRUE, FALSE	-
minsize	integer	-		$[1, \infty)$
minsplit	integer	-		$[1, \infty)$
minbucket	integer	-		$[1, \infty)$
maxdepth	integer	Inf		$[0, \infty)$
mtry	integer	Inf		$[0, \infty)$
trim	numeric	0.1		$[0, \infty)$
breakties	logical	FALSE	TRUE, FALSE	-
parm	untyped	-		-
dfsplit	integer	-		$[0, \infty)$
prune	untyped	-		-
restart	logical	TRUE	TRUE, FALSE	-
verbose	logical	FALSE	TRUE, FALSE	-
caseweights	logical	TRUE	TRUE, FALSE	-
ytype	character	vector	vector, matrix, data.frame	-
xtype	character	matrix	vector, matrix, data.frame	-

terminal	untyped	"object"		-
inner	untyped	"object"		-
model	logical	TRUE	TRUE, FALSE	-
numsplit	character	left	left, center	-
catsplit	character	binary	binary, multiway	-
vcov	character	opg	opg, info, sandwich	-
ordinal	character	chisq	chisq, max, L2	-
nrep	integer	10000		$[0, \infty)$
applyfun	untyped	-		-
cores	integer	NULL		$(-\infty, \infty)$
additional	untyped	-		-
predict_fun	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifMob`

Methods

Public methods:

- `LearnerClassifMob$new()`
- `LearnerClassifMob$clone()`

`LearnerClassifMob$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifMob$new()
```

`LearnerClassifMob$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifMob$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sumny

References

Hothorn T, Zeileis A (2015). "partykit: A Modular Toolkit for Recursive Partytioning in R." *Journal of Machine Learning Research*, **16**(118), 3905-3909. http://jmlr.org/papers/v16/hothorn15a.html.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. doi:10.1198/106186006x133933. https://doi.org/10.1198/106186006x133933.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
library(mlr3)
logit_ = function(y, x, start = NULL, weights = NULL, offset = NULL, ...) {
  glm(y ~ 1, family = binomial, start = start, ...)
}
learner = LearnerClassifMob$new()
learner$param_set$values$rhs = "."
learner$param_set$values$fit = logit_
learner$param_set$values$additional = list(maxit = 100)
learner$feature_types = c("logical", "integer", "numeric", "factor", "ordered")
learner$properties = c("twoclass", "weights")

predict_fun = function(object, newdata, task, .type) {
  p = unname(predict(object, newdata = newdata, type = "response"))
  levs = task$levels(task$target_names)[[1L]]

  if (.type == "response") {
    ifelse(p < 0.5, levs[1L], levs[2L])
  } else {
    y = matrix(c(1 - p, p), ncol = 2L, nrow = length(p))
    colnames(y) = levs
    y
  }
}
task = tsk("breast_cancer")
learner$param_set$values$predict_fun = predict_fun
ids = partition(task)
learner$train(task, row_ids = ids$train)
learner$predict(task, row_ids = ids$test)
```

`mlr_learners_classif.multilayer_perceptron`
Classification MultilayerPerceptron Learner

Description

Classifier that uses backpropagation to learn a multi-layer perceptron. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern
- `G` removed:
  - GUI will be opened
- Reason for change: The parameter is removed because we don't want to launch GUI.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.multilayer_perceptron")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
L	numeric	0.3		$[0, 1]$
M	numeric	0.2		$[0, 1]$
N	integer	500		$[1, \infty)$
V	numeric	0		$[0, 100]$
S	integer	0		$[0, \infty)$
E	integer	20		$[1, \infty)$
A	logical	FALSE	TRUE, FALSE	-
B	logical	FALSE	TRUE, FALSE	-
H	untyped	"a"		-
C	logical	FALSE	TRUE, FALSE	-
I	logical	FALSE	TRUE, FALSE	-
R	logical	FALSE	TRUE, FALSE	-
D	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifMultilayerPerceptron`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifMultilayerPerceptron$new()`
- `LearnerClassifMultilayerPerceptron$marshal()`
- `LearnerClassifMultilayerPerceptron$unmarshal()`
- `LearnerClassifMultilayerPerceptron$clone()`

`LearnerClassifMultilayerPerceptron$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifMultilayerPerceptron$new()
```

`LearnerClassifMultilayerPerceptron$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifMultilayerPerceptron$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifMultilayerPerceptron$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifMultilayerPerceptron$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifMultilayerPerceptron$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifMultilayerPerceptron$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.multilayer_perceptron")
print(learner)

# Define a Task
```

```
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.naive_bayes_multinomial`
Classification Multinomial Naive Bayes Learner From Weka

Description

Multinomial Naive Bayes classifier. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.naive_bayes_multinomial")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifNaiveBayesMultinomial`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifNaiveBayesMultinomial$new()`
- `LearnerClassifNaiveBayesMultinomial$marshal()`
- `LearnerClassifNaiveBayesMultinomial$unmarshal()`
- `LearnerClassifNaiveBayesMultinomial$clone()`

`LearnerClassifNaiveBayesMultinomial$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifNaiveBayesMultinomial$new()
```

`LearnerClassifNaiveBayesMultinomial$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifNaiveBayesMultinomial$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifNaiveBayesMultinomial$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifNaiveBayesMultinomial$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifNaiveBayesMultinomial$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifNaiveBayesMultinomial$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Mccallum A, Nigam K (1998). "A Comparison of Event Models for Naive Bayes Text Classification." In *AAAI-98 Workshop on 'Learning for Text Categorization'*.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.naive_bayes_multinomial")
print(learner)

# Define a Task
```

```
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.naive_bayes_weka`
Classification Naive Bayes Learner From Weka

Description

Naive Bayes Classifier Using Estimator Classes. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.naive_bayes_weka")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
K	logical	FALSE	TRUE, FALSE	-
D	logical	FALSE	TRUE, FALSE	-
O	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifNaiveBayesWeka`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifNaiveBayesWeka$new()`
- `LearnerClassifNaiveBayesWeka$marshal()`
- `LearnerClassifNaiveBayesWeka$unmarshal()`
- `LearnerClassifNaiveBayesWeka$clone()`

`LearnerClassifNaiveBayesWeka$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifNaiveBayesWeka$new()
```

`LearnerClassifNaiveBayesWeka$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifNaiveBayesWeka$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifNaiveBayesWeka$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifNaiveBayesWeka$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifNaiveBayesWeka$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifNaiveBayesWeka$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

John GH, Langley P (1995). "Estimating Continuous Distributions in Bayesian Classifiers." In *Eleventh Conference on Uncertainty in Artificial Intelligence*, 338-345.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.naive_bayes_weka")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.neuralnet

Classification Neural Network Learner

Description

Calls `neuralnet::neuralnet()` from **neuralnet**.

Initial parameter values

- `err.fct` is set to `"ce"`.
- `linear.output` is set to `FALSE` to perform classification.
- `act.fct` is set to `"logistic"` for classification.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.neuralnet")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **neuralnet**

Parameters

Id	Type	Default	Levels	Range
act.fct	untyped	"logistic"		-
algorithm	character	rprop+	backprop, rprop+, rprop-, sag, slr	-
constant.weights	untyped	-		-
err.fct	character	ce	sse, ce	-
exclude	untyped	-		-
hidden	untyped	1L		-
learningrate	numeric	-		$(-\infty, \infty)$
learningrate.factor	untyped	list(minus = 0.5, plus = 1.2)		-
learningrate.limit	untyped	-		-
lifesign	character	none	none, minimal, full	-
lifesign.step	integer	1000		$(-\infty, \infty)$
likelihood	logical	FALSE	TRUE, FALSE	-
linear.output	logical	FALSE	TRUE, FALSE	-
rep	integer	1		$[1, \infty)$
startweights	untyped	-		-
stepmax	numeric	1e+05		$(-\infty, \infty)$
threshold	numeric	0.01		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifNeuralnet`

Methods

Public methods:

- `LearnerClassifNeuralnet$new()`
- `LearnerClassifNeuralnet$clone()`

`LearnerClassifNeuralnet$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifNeuralnet$new()
```

`LearnerClassifNeuralnet$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifNeuralnet$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.neuralnet")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.nnTrain`
Classification Neural Network Learner

Description

Calls `deepnet::nn.train()` from **deepnet**.

Initial parameter values

- `output` is set to `"softmax"` for probabilistic classification.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.nnTrain")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **deepnet**

Parameters

Id	Type	Default	Levels	Range
activationfun	character	sigm	sigm, linear, tanh	-
batchsize	integer	100		$[1, \infty)$
hidden	untyped	10L		-
hidden_dropout	numeric	0		$[0, 1]$
initB	untyped	-		-
initW	untyped	-		-
learningrate	numeric	0.8		$[0, \infty)$
learningrate_scale	numeric	1		$[0, \infty)$
max_number_of_layers	integer	-		$[1, \infty)$
momentum	numeric	0.5		$[0, \infty)$
numepochs	integer	3		$[1, \infty)$
output	character	-	sigm, linear, softmax	-
visible_dropout	numeric	0		$[0, 1]$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifNNTrain`

Methods

Public methods:

- `LearnerClassifNNTrain$new()`
- `LearnerClassifNNTrain$clone()`

`LearnerClassifNNTrain$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifNNTrain$new()
```

`LearnerClassifNNTrain$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifNNTrain$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Rong, Xiao (2022). "deepnet: Deep Learning Toolkit in R." *R package version 0.2.1*. doi:10.32614/CRAN.package.deepnet. https://CRAN.R-project.org/package=deepnet.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.nnTrain")
print(learner)

# Define a Task
task = tsk("sonar")
```

```
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.OneR`

Classification OneR Learner

Description

One Rule classification algorithm that yields an extremely simple model. Calls `RWeka::OneR()` from **RWeka**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.OneR")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
B	integer	6		$[1, \infty)$
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$

batch_size	integer	100	$[1, \infty)$
options	untyped	NULL	-

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifOneR`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifOneR$new()`
- `LearnerClassifOneR$marshal()`
- `LearnerClassifOneR$unmarshal()`
- `LearnerClassifOneR$clone()`

`LearnerClassifOneR$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifOneR$new()
```

`LearnerClassifOneR$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifOneR$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifOneR$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifOneR$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifOneR$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifOneR$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

henrifnk

References

Holte, C R (1993). "Very simple classification rules perform well on most commonly used datasets." *Machine learning*, **11**(1), 63–90.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.OneR")
print(learner)

# Define a Task
task = tsk("sonar")
```

```
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.opls`
OPLS-DA Classification Learner

Description

Binary classification with orthogonal partial least squares discriminant analysis (OPLS-DA). Calls `ropls::opls()` from Bioconductor package **ropls**.

`ropls` is distributed via Bioconductor and not via CRAN, so it has to be installed with `BiocManager::install("ropls")`.

Initial parameter values

- `predI`
  - Actual default: `NA`, which lets `ropls::opls()` determine the number of predictive components by cross-validation.
  - Adjusted default: `1`
  - Reason for change: OPLS-DA is defined with a single predictive component, and a fixed number of components keeps the model comparable across resampling iterations.
- `orthoI`
  - Actual default: `0`
  - Adjusted default: `1`
  - Reason for change: With `orthoI = 0` no orthogonal component is extracted and `ropls::opls()` fits plain PLS-DA instead of OPLS-DA.
- `permI`
  - Actual default: `20`
  - Adjusted default: `0`
  - Reason for change: The permutation test is a model diagnostic that is not used for prediction and costs `permI` extra model fits.

Suppressed output

`ropls::opls()` writes a diagnostic figure and a model summary by default. Both are suppressed by fixing `fig.pdfC` and `info.txtC` to `"none"`, so neither is exposed as a hyperparameter.

Number of components

`ropls::opls()` requires `predI + orthoI` to not exceed the smaller of the number of observations and the number of features, and errors otherwise. With the initial parameter values this means that the learner needs at least two features.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.opls")
```

Meta Information

- Task type: "classif"
- Predict Types: "response"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **ropls**

Parameters

Id	Type	Default	Levels	Range
predI	integer	-		$[1, \infty)$
orthoI	integer	0		$[0, \infty)$
algoC	character	default	default, nipals, svd	-
crossvalI	integer	7		$[1, \infty)$
log10L	logical	FALSE	TRUE, FALSE	-
scaleC	character	standard	none, center, pareto, standard	-
permI	integer	20		$[0, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifOPLS`

Methods

Public methods:

- `LearnerClassifOPLS$new()`
- `LearnerClassifOPLS$clone()`

`LearnerClassifOPLS$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifOPLS$new()
```

`LearnerClassifOPLS$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifOPLS$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Takha65

References

Thévenot, A. E, Roux, Aurélie, Xu, Ying, Ezan, Eric, Junot, Christophe (2015). "Analysis of the Human Adult Urinary Metabolome Variations with Age, Body Mass Index, and Gender by Implementing a Comprehensive Workflow for Univariate and OPLS Statistical Analyses." *Journal of Proteome Research*, **14**(8), 3322–3335. doi:10.1021/acs.jproteome.5b00354.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.opls")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
```

```
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.PART`
Classification PART Learner

Description

Regression partition tree. Calls `RWeka::PART()` from **RWeka**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.PART")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
C	numeric	0.25		$[2.22044604925031e-16, 1]$
M	integer	2		$[1, \infty)$
R	logical	FALSE	TRUE, FALSE	-
N	integer	3		$[1, \infty)$
B	logical	FALSE	TRUE, FALSE	-
U	logical	FALSE	TRUE, FALSE	-
J	logical	FALSE	TRUE, FALSE	-
Q	integer	1		$[1, \infty)$

doNotMakeSplitPointActualValue	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifPART`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifPART$new()`
- `LearnerClassifPART$marshal()`
- `LearnerClassifPART$unmarshal()`
- `LearnerClassifPART$clone()`

`LearnerClassifPART$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifPART$new()
```

`LearnerClassifPART$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifPART$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifPART$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifPART$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifPART$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifPART$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

henrifnk

References

Frank, Eibe, Witten, H I (1998). "Generating accurate rule sets without global optimization."

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.PART")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.penalized

Classification L1 and L2 Penalized Learner

Description

Penalized (L1 and L2) Logistic Regression. Calls `penalized::penalized()` from: **penalized**.

Details

The `penalized` and `unpenalized` arguments in the learner are implemented slightly differently than in `penalized::penalized()`. Here, there is no parameter for `penalized` but instead it is assumed that every variable is penalized unless stated in the `unpenalized` parameter.

Initial parameter values

- `trace` is set to `"FALSE"` to disable printing output during model training.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.penalized")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **penalized**

Parameters

Id	Type	Default	Levels	Range
epsilon	numeric	1e-10		$[0, \infty)$
fusedl	logical	FALSE	TRUE, FALSE	-
lambda1	numeric	0		$[0, \infty)$
lambda2	numeric	0		$[0, \infty)$
maxiter	integer	-		$[1, \infty)$
positive	untyped	FALSE		-
standardize	logical	FALSE	TRUE, FALSE	-
startbeta	untyped	-		-
startgamma	untyped	-		-
steps	untyped	1L		-
trace	logical	TRUE	TRUE, FALSE	-
unpenalized	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifPenalized`

Methods

Public methods:

- `LearnerClassifPenalized$new()`
- `LearnerClassifPenalized$clone()`

`LearnerClassifPenalized$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifPenalized$new()
```

`LearnerClassifPenalized$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifPenalized$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Goeman, J J (2010). "L1 penalized estimation in the Cox proportional hazards model." *Biometrical journal*, **52**(1), 70–84.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.penalized")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.plsdaCaret`
Partial Least Squares Discriminant Analysis Learner

Description

Partial Least Squares Discriminant Analysis for classification. Calls `caret::plsda()` from **caret**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.plsdaCaret")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **caret**, **pls**

Parameters

Id	Type	Default	Levels	Range
ncomp	integer	2		$[1, \infty)$
prior	untyped	"softmax"		-
probMethod	character	softmax	softmax, Bayes	-
method	character	kernelpls	kernelpls, widekernelpls, simpls, oscorespls	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifPlsdaCaret`

Methods

Public methods:

- `LearnerClassifPlsdaCaret$new()`
- `LearnerClassifPlsdaCaret$clone()`

`LearnerClassifPlsdaCaret$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifPlsdaCaret$new()
```

`LearnerClassifPlsdaCaret$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifPlsdaCaret$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Kuhn, Max (2008). "Building Predictive Models in R Using the caret Package." *Journal of Statistical Software*, **28**(5), 1–26. doi:10.18637/jss.v028.i05.

Mevik, Bjorn-Helge, Wehrens, Ron (2007). "The pls Package: Principal Component and Partial Least Squares Regression in R." *Journal of Statistical Software*, **18**(2), 1–24. doi:10.18637/jss.v018.i02.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.plsdaCaret")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.polyFit` *Polynomial GLM Learner*

Description

Polynomial GLM. Calls `polyreg::polyFit()` from **polyreg**.

Initial parameter values

- `deg`: We have set this to 2, pretty arbitrarily.
- `noisy`: We have set this to FALSE, to get no output on the console.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.polyFit")
```

Meta Information

- Task type: "classif"
- Predict Types: "response"
- Feature Types: "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **polyreg**

Parameters

Id	Type	Default	Levels	Range
deg	integer	-		$[0, \infty)$
maxInteractDeg	integer	-		$[0, \infty)$
return_xy	logical	FALSE	TRUE, FALSE	-
returnPoly	logical	FALSE	TRUE, FALSE	-
noisy	logical	TRUE	TRUE, FALSE	-
glmMethod	character	one	one, all	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifPolyFit`

Methods

Public methods:

- `LearnerClassifPolyFit$new()`
- `LearnerClassifPolyFit$clone()`

`LearnerClassifPolyFit$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifPolyFit$new()
```

`LearnerClassifPolyFit$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifPolyFit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

`mlr_learners_classif.priority_lasso`

Classification Priority Lasso Learner

Description

Patient outcome prediction based on multi-omics data, taking practitioners' preferences into account. Calls `prioritylasso::prioritylasso()` from **prioritylasso**. Only binary classification is supported.

Scope and supported arguments

This learner intentionally exposes a focused subset of training and prediction arguments. It is designed to work well out of the box, without requiring extensive parameter tuning. Some arguments from `cv.glmnet()`, `glmnet()`, and `predict.prioritylasso()` are not included, because they are not consistently supported or forwarded through the full train/predict path (e.g. handling missing test data, or performing relaxed lasso fits).

Please open an issue if there is a need for supporting more learner parameters.

Initial parameter values

- `family` is set to `"binomial"` and cannot be changed

Custom mlr3 parameters

- `adaptive.order`: if `TRUE`, the priority order of blocks is estimated from the data following Herrmann et al. (2021), instead of using the user-supplied block order. For each block, a Ridge regression (`alpha = 0`) is fit using `cv.glmnet()` on that block alone. The importance of a block is measured by the mean absolute coefficient (MAC) score at the `lambda.min` value (the lambda giving minimum cross-validation error). A penalty factor of `1 / MAC` is then assigned to each block. Blocks are sorted by increasing penalty factor, i.e., blocks with larger MAC (stronger average signal) receive higher priority (come first). Also, the block-wise penalty factors are attached to the fitted model object as `learner$model$block.penalty.factors`.

This method is useful when no domain knowledge is available to specify block priority. In this step, data are **standardized by default** (`standardize = TRUE`), but this can be overridden by the learner's `standardize` parameter. `lambda.min` is always used to derive the block priority. Additional arguments such as `nfolds`, `type.measure`, `weights` and `cox.ties` (if provided) are forwarded to each block-wise `cv.glmnet()` fit. The `max.coef` parameter, if supplied, it is re-ordered accordingly to align with the new block order.

This parameter is ignored when fewer than two blocks are provided. It defaults to `FALSE` for backward compatibility.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.priority_lasso")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **prioritylasso**

Parameters

Id	Type	Default	Levels	Range
type.measure	character	-	class, auc	-
blocks	untyped	-		-
max.coef	untyped	NULL		-
block1.penalization	logical	TRUE	TRUE, FALSE	-
lambda.type	character	lambda.min	lambda.min, lambda.1se	-
standardize	logical	TRUE	TRUE, FALSE	-
nfolds	integer	10		$[3, \infty)$
foldid	untyped	NULL		-
cvoffset	logical	FALSE	TRUE, FALSE	-
cvoffsetnfolds	integer	10		$[1, \infty)$
return.x	logical	TRUE	TRUE, FALSE	-
lambda	untyped	NULL		-
grouped	logical	TRUE	TRUE, FALSE	-
trace.it	integer	0		$[0, 1]$
type.logistic	character	Newton	Newton, modified.Newton	-
include.allintercepts	logical	FALSE	TRUE, FALSE	-
use.blocks	untyped	"all"		-
adaptive.order	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifPriorityLasso`

Methods

Public methods:

- `LearnerClassifPriorityLasso$new()`
- `LearnerClassifPriorityLasso$selected_features()`
- `LearnerClassifPriorityLasso$clone()`

`LearnerClassifPriorityLasso$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifPriorityLasso$new()
```

`LearnerClassifPriorityLasso$selected_features()`: Selected features, i.e. those where the coefficient is non-zero.

Usage:

```
LearnerClassifPriorityLasso$selected_features()
```

Returns: `character()`.

`LearnerClassifPriorityLasso$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifPriorityLasso$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

HarutyunyanLiana

bblodfon

References

Simon K, Vindi J, Roman H, Tobias H, Anne-Laure B (2018). "Priority-Lasso: a simple hierarchical approach to the prediction of clinical outcome using multi-omics data." *BMC Bioinformatics*, **19**. doi:10.1186/s1285901823446.

Herrmann, M., Probst, P., Hornung, R., Jurinovic, V., Boulesteix, L. A (2021). "Large-scale benchmark study of survival prediction methods using multi-omics data." *Briefings in Bioinformatics*, **22**(3), 1–15. doi:10.1093/BIB/BBAA167.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define a Task
task = mlr3::as_task_classif(prioritylasso::pl_data, target = "pl_out")

# Create train and test set
ids = partition(task)

# Define the Learner and set parameter values
learner = lrn("classif.priority_lasso", type.measure = "auc", standardize = FALSE,
  blocks = list(bp1 = 1:4, bp2 = 5:9, bp3 = 10:28, bp4 = 29:1028),
  max.coef = c(Inf, Inf, 10, 10))
print(learner)

# Train the learner
learner$train(task, row_ids = ids$train)
```

```
# Selected features
learner$selected_features()

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.random_forest_weka`
Classification Random Forest Learner from Weka

Description

Class for constructing a random forest. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- `store_out_of_bag_predictions`:
  - original id: store-out-of-bag-predictions
- `output_out_of_bag_complexity_statistics`:
  - original id: output-out-of-bag-complexity-statistics
- `num_slots`:
  - original id: num-slots
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern
- `attribute-importance` removed:
  - Compute and output attribute importance (mean impurity decrease method)
- Reason for change: The parameter is removed because it's unclear how to actually use it.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.random_forest_weka")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
P	numeric	100		$[0, 100]$
O	logical	FALSE	TRUE, FALSE	-
store_out_of_bag_predictions	logical	FALSE	TRUE, FALSE	-
output_out_of_bag_complexity_statistics	logical	FALSE	TRUE, FALSE	-
print	logical	FALSE	TRUE, FALSE	-
I	integer	100		$[1, \infty)$
num_slots	integer	1		$(-\infty, \infty)$
K	integer	0		$(-\infty, \infty)$
M	integer	1		$[1, \infty)$
V	numeric	0.001		$(-\infty, \infty)$
S	integer	1		$(-\infty, \infty)$
depth	integer	0		$[0, \infty)$
N	integer	0		$(-\infty, \infty)$
U	logical	FALSE	TRUE, FALSE	-
B	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifRandomForestWeka`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifRandomForestWeka$new()`
- `LearnerClassifRandomForestWeka$marshal()`
- `LearnerClassifRandomForestWeka$unmarshal()`
- `LearnerClassifRandomForestWeka$clone()`

`LearnerClassifRandomForestWeka$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifRandomForestWeka$new()
```

`LearnerClassifRandomForestWeka$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifRandomForestWeka$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifRandomForestWeka$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifRandomForestWeka$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifRandomForestWeka$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifRandomForestWeka$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Breiman, Leo (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565. doi:10.1023/A:1010933404324.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.random_forest_weka")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.random_tree`

Classification Random Tree Learner

Description

Tree that considers K randomly chosen attributes at each node. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.random_tree")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
K	integer	0		$[0, \infty)$
M	integer	1		$[1, \infty)$
V	numeric	0.001		$(-\infty, \infty)$
S	integer	1		$(-\infty, \infty)$
depth	integer	0		$[0, \infty)$
N	integer	0		$[0, \infty)$
U	logical	FALSE	TRUE, FALSE	-
B	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifRandomTree`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifRandomTree$new()`
- `LearnerClassifRandomTree$marshal()`
- `LearnerClassifRandomTree$unmarshal()`
- `LearnerClassifRandomTree$clone()`

`LearnerClassifRandomTree$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifRandomTree$new()
```

`LearnerClassifRandomTree$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifRandomTree$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifRandomTree$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifRandomTree$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifRandomTree$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifRandomTree$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.random_tree")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.randomForest`

Classification Random Forest Learner

Description

Random forest for classification. Calls `randomForest::randomForest()` from **randomForest**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.randomForest")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **randomForest**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
mtry	integer	-		$[1, \infty)$
replace	logical	TRUE	TRUE, FALSE	-
classwt	untyped	NULL		-
cutoff	untyped	-		-
strata	untyped	-		-
sampsize	untyped	-		-
nodesize	integer	1		$[1, \infty)$
maxnodes	integer	-		$[1, \infty)$
importance	character	FALSE	accuracy, gini, none	-
localImp	logical	FALSE	TRUE, FALSE	-
proximity	logical	FALSE	TRUE, FALSE	-
oob.prox	logical	-	TRUE, FALSE	-
norm.votes	logical	TRUE	TRUE, FALSE	-
do.trace	logical	FALSE	TRUE, FALSE	-
keep.forest	logical	TRUE	TRUE, FALSE	-
keep.inbag	logical	FALSE	TRUE, FALSE	-
predict.all	logical	FALSE	TRUE, FALSE	-
nodes	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifRandomForest`

Methods

Public methods:

- `LearnerClassifRandomForest$new()`

- `LearnerClassifRandomForest$importance()`
- `LearnerClassifRandomForest$oob_error()`
- `LearnerClassifRandomForest$clone()`

`LearnerClassifRandomForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifRandomForest$new()
```

`LearnerClassifRandomForest$importance()`: The importance scores are extracted from the slot `importance`. Parameter 'importance' must be set to either `"accuracy"` or `"gini"`.

Usage:

```
LearnerClassifRandomForest$importance()
```

Returns: Named `numeric()`.

`LearnerClassifRandomForest$oob_error()`: OOB errors are extracted from the model slot `err.rate`.

Usage:

```
LearnerClassifRandomForest$oob_error()
```

Returns: `numeric(1)`.

`LearnerClassifRandomForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifRandomForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

pat-s

References

Breiman, Leo (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565. doi:10.1023/A:1010933404324.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.randomForest", importance = "accuracy")
print(learner)

# Define a Task
task = tsk("sonar")
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.rda

Classification Discriminant Analysis Learner

Description

Regularized Discriminant Analysis. Calls `klaR::rda()` from: **klaR**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.rda")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **klaR**

Parameters

Id	Type	Default	Levels	Range
lambda	numeric	-		$[0, 1]$
gamma	numeric	-		$[0, 1]$
crossval	logical	TRUE	TRUE, FALSE	-
fold	integer	10		$[1, \infty)$
train.fraction	numeric	0.5		$[0, 1]$
output	logical	FALSE	TRUE, FALSE	-
schedule	integer	2		$(-\infty, \infty)$
T.start	numeric	0.1		$[0, \infty)$
halflife	numeric	50		$[0, \infty)$
zero.temp	numeric	0.01		$[0, \infty)$
alpha	numeric	2		$[1, \infty)$
K	integer	100		$[1, \infty)$
trafo	logical	TRUE	TRUE, FALSE	-
simAnn	logical	FALSE	TRUE, FALSE	-
estimate.error	logical	TRUE	TRUE, FALSE	-
max.iter	integer	100		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifRda`

Methods

Public methods:

- `LearnerClassifRda$new()`
- `LearnerClassifRda$clone()`

`LearnerClassifRda$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifRda$new()
```

`LearnerClassifRda$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifRda$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Friedman JH (1989). "Regularized Discriminant Analysis." *Journal of the American Statistical Association*, **84**(405), 165–175. ISSN 01621459, 1537274X. http://www.jstor.org/stable/2289860.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.rda")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.reptree`
Classification Decision Tree Learner

Description

Fast decision tree learner. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.reptree")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
M	integer	2		$(-\infty, \infty)$
V	numeric	0.001		$(-\infty, \infty)$
N	integer	3		$(-\infty, \infty)$
S	integer	1		$(-\infty, \infty)$

P	logical	-	TRUE, FALSE	-
L	integer	-1		$(-\infty, \infty)$
I	integer	0		$(-\infty, \infty)$
R	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifREPTree`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifREPTree$new()`
- `LearnerClassifREPTree$marshal()`
- `LearnerClassifREPTree$unmarshal()`
- `LearnerClassifREPTree$clone()`

`LearnerClassifREPTree$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifREPTree$new()
```

`LearnerClassifREPTree$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifREPTree$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifREPTree$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifREPTree$unmarshal(...)
```

Arguments:

... (any)
Additional arguments passed to mlr3::unmarshal_model().

LearnerClassifREPTree$clone(): The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifREPTree$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.reptree")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.rFerns`

Random Ferns Classification Learner

Description

Ensemble machine learning algorithm based on Random Ferns, which are a simplified, faster alternative to Random Forests. Calls `rFerns::rFerns()` from **rFerns**.

Initial parameter values

- `importance`:
  - Actual default: `FALSE`
  - Initial value: `"simple"`
  - Reason for change: The default value of `FALSE` will resolve to "none", which turns importance calculation off. To enable importance calculation by default, `importance` is set to `"simple"`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.rFerns")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”
- Feature Types: “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3extralearners**, **rFerns**

Parameters

Id	Type	Default	Levels	Range
consistentSeed	untyped	NULL		-
depth	integer	5		$[1, 16]$
ferns	integer	1000		$(-\infty, \infty)$
importance	untyped	FALSE		-
saveForest	logical	TRUE	TRUE, FALSE	-
threads	integer	0		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifRferns`

Methods

Public methods:

- `LearnerClassifRferns$new()`
- `LearnerClassifRferns$importance()`
- `LearnerClassifRferns$oob_error()`
- `LearnerClassifRferns$clone()`

`LearnerClassifRferns$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifRferns$new()
```

`LearnerClassifRferns$importance()`: The importance scores are extracted from the model slot `importance`.

Usage:

```
LearnerClassifRferns$importance()
```

Returns: Named `numeric()`.

`LearnerClassifRferns$oob_error()`: OOB error is extracted from the model slot `oobErr`.

Usage:

```
LearnerClassifRferns$oob_error()
```

Returns: `numeric(1)`.

`LearnerClassifRferns$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifRferns$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Kursa MB (2014). "rFerns: An Implementation of the Random Ferns Method for General-Purpose Machine Learning." *Journal of Statistical Software*, **61**(10), 1–13. https://www.jstatsoft.org/v61/i10/.

Ozuysal, Mustafa, Calonder, Michael, Lepetit, Vincent, Fua, Pascal (2010). "Fast Keypoint Recognition Using Random Ferns." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **32**(3), 448-461. doi:10.1109/TPAMI.2009.23.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.rFerns")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.rfsrc`

Classification Random Forest SRC Learner

Description

Random forest for classification. Calls `randomForestSRC::rfsrc()` from **randomForestSRC**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.rfsrc")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3extralearners**, **randomForestSRC**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
mtry	integer	-		$[1, \infty)$
mtry.ratio	numeric	-		$[0, 1]$
nodesize	integer	15		$[1, \infty)$
nodedepth	integer	-		$[1, \infty)$
splitrule	character	gini	gini, auc, entropy	-
nsplit	integer	10		$[0, \infty)$
importance	character	FALSE	FALSE, TRUE, none, permute, random, anti	-
block.size	integer	10		$[1, \infty)$
bootstrap	character	by.root	by.root, by.node, none, by.user	-
samptype	character	swor	swor, swr	-
samp	untyped	-		-
membership	logical	FALSE	TRUE, FALSE	-
sampsize	untyped	-		-
sampsize.ratio	numeric	-		$[0, 1]$
na.action	character	na.omit	na.omit, na.impute	-
nimpute	integer	1		$[1, \infty)$
proximity	character	FALSE	FALSE, TRUE, inbag, oob, all	-
distance	character	FALSE	FALSE, TRUE, inbag, oob, all	-
forest.wt	character	FALSE	FALSE, TRUE, inbag, oob, all	-
xvar.wt	untyped	-		-
split.wt	untyped	-		-
forest	logical	TRUE	TRUE, FALSE	-
var.used	character	FALSE	FALSE, all.trees	-
split.depth	character	FALSE	FALSE, all.trees, by.tree	-
seed	integer	-		$(-\infty, -1]$
do.trace	logical	FALSE	TRUE, FALSE	-
get.tree	untyped	-		-
outcome	character	train	train, test	-
ptn.count	integer	0		$[0, \infty)$
cores	integer	1		$[1, \infty)$
save.memory	logical	FALSE	TRUE, FALSE	-

perf.type	character	-	gmean, misclass, brier, none	-
case.depth	logical	FALSE	TRUE, FALSE	-
marginal.xvar	untyped	NULL		-

Custom mlr3 parameters

- `mtry`: This hyperparameter can alternatively be set via the added hyperparameter `mtry.ratio` as `mtry = max(ceiling(mtry.ratio * n_features), 1)`. Note that `mtry` and `mtry.ratio` are mutually exclusive.
- `sampsize`: This hyperparameter can alternatively be set via the added hyperparameter `sampsize.ratio` as `sampsize = max(ceiling(sampsize.ratio * n_obs), 1)`. Note that `sampsize` and `sampsize.ratio` are mutually exclusive.
- `cores`: This value is set as the option `rf.cores` during training and is set to 1 by default.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifRandomForestSRC`

Methods

Public methods:

- `LearnerClassifRandomForestSRC$new()`
- `LearnerClassifRandomForestSRC$importance()`
- `LearnerClassifRandomForestSRC$selected_features()`
- `LearnerClassifRandomForestSRC$oob_error()`
- `LearnerClassifRandomForestSRC$clone()`

`LearnerClassifRandomForestSRC$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifRandomForestSRC$new()
```

`LearnerClassifRandomForestSRC$importance()`: The importance scores are extracted from the model slot `importance`, returned for 'all'.

Usage:

```
LearnerClassifRandomForestSRC$importance()
```

Returns: Named `numeric()`.

`LearnerClassifRandomForestSRC$selected_features()`: Selected features are extracted from the model slot `var.used`.

Note: Due to a known issue in `randomForestSRC`, enabling `var.used = "all.trees"` causes prediction to fail. Therefore, this setting should be used exclusively for feature selection purposes and not when prediction is required.

Usage:

`LearnerClassifRandomForestSRC$selected_features()`

Returns: `character()`.

`LearnerClassifRandomForestSRC$oob_error()`: OOB error extracted from the model slot `err.rate`.

Usage:

`LearnerClassifRandomForestSRC$oob_error()`

Returns: `numeric()`.

`LearnerClassifRandomForestSRC$clone()`: The objects of this class are cloneable with this method.

Usage:

`LearnerClassifRandomForestSRC$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Breiman, Leo (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565. doi:10.1023/A:1010933404324.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.rfsrc", importance = "TRUE")
print(learner)

# Define a Task
task = tsk("sonar")
```

```
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.rotationForest`

Classification Rotation Forest Learner

Description

Classification using Rotation Forest algorithm. Training this learner works only for tasks with at least two features. The task must not have utf8 feature names. Calls `rotationForest::rotationForest()` from **rotationForest**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.rotationForest")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **rotationForest**

Parameters

Id	Type	Default	Range
K	integer	3	$[1, \infty)$
L	integer	10	$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifRotationForest`

Methods

Public methods:

- `LearnerClassifRotationForest$new()`
- `LearnerClassifRotationForest$clone()`

`LearnerClassifRotationForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifRotationForest$new()
```

`LearnerClassifRotationForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifRotationForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Rodriguez, J.J., Kuncheva, L.I., Alonso, C.J. (2006). "Rotation Forest: A New Classifier Ensemble Method." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **28**(10), 1619-1630. doi:10.1109/TPAMI.2006.211.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.rotationForest")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.saeDNN

Classification Stacked Autoencoder Deep Neural Network Learner

---

Description

Calls `deepnet::sae.dnn.train()` from **deepnet**.

Initial parameter values

- `output` is set to `"softmax"` for probabilistic classification.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.saeDNN")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “integer”, “numeric”
- Required Packages: **mlr3**, **deepnet**

Parameters

Id	Type	Default	Levels	Range
hidden	untyped	10L		-
activationfun	character	sigm	sigm, linear, tanh	-
learningrate	numeric	0.8		$[0, \infty)$
momentum	numeric	0.5		$[0, \infty)$
learningrate_scale	numeric	1		$[0, \infty)$
numepochs	integer	3		$[1, \infty)$
batchsize	integer	100		$[1, \infty)$
output	character	-	sigm, linear, softmax	-
sae_output	character	linear	sigm, linear, softmax	-
hidden_dropout	numeric	0		$[0, 1]$
visible_dropout	numeric	0		$[0, 1]$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifSaeDNN`

Methods

Public methods:

- `LearnerClassifSaeDNN$new()`
- `LearnerClassifSaeDNN$clone()`

`LearnerClassifSaeDNN$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifSaeDNN$new()
```

`LearnerClassifSaeDNN$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifSaeDNN$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Rong, Xiao (2022). "deepnet: Deep Learning Toolkit in R." *R package version 0.2.1*. doi:10.32614/CRAN.package.deepnet. https://CRAN.R-project.org/package=deepnet.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.saeDNN")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.sda`

Shrinkage Discriminant Analysis

Description

Shrinkage Discriminant Analysis for classification. Calls `sda::sda()` from **sda**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.sda")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **sda**

Parameters

Id	Type	Default	Levels	Range
lambda	numeric	-		$[0, 1]$
lambda.var	numeric	-		$[0, 1]$
lambda.freqs	numeric	-		$[0, 1]$
diagonal	logical	FALSE	TRUE, FALSE	-
verbose	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifSda`

Methods

Public methods:

- `LearnerClassifSda$new()`
- `LearnerClassifSda$clone()`

`LearnerClassifSda$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifSda$new()
```

`LearnerClassifSda$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifSda$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

annanzrv

References

Ahdesmaeki, Miika, Strimmer, Korbinian (2010). "Feature selection in omics prediction problems using cat scores and false nondiscovery rate control." *The Annals of Applied Statistics*, **4**(1). ISSN 1932-6157. doi:10.1214/09aoas277. http://dx.doi.org/10.1214/09-AOAS277.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.sda")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.sdlda` *Classification Linear Discriminant Analysis Learner*

Description

Shrinkage-based Diagonal Linear Discriminant Analysis classifier. Type of Naive Bayes classifiers that improves the estimation of the pooled variances by using a shrinkage-based estimator of the pooled covariance matrix. Calls `sparsediscrim::lda_shrink_cov()` **sparsediscrim**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.sdlda")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **sparsediscrim**

Parameters

Id	Type	Default	Range
num_alphas	integer	101	$(-\infty, \infty)$
prior	untyped	NULL	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifSdlda`

Methods

Public methods:

- `LearnerClassifSdlda$new()`
- `LearnerClassifSdlda$clone()`

`LearnerClassifSdlda$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifSdlda$new()
```

`LearnerClassifSdlda$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifSdlda$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Pang H, Tong T, Zhao H (2009). "Shrinkage-Based Diagonal Discriminant Analysis and Its Applications in High-Dimensional Data." *Biometrics*, **65**(4), 1021–1029. ISSN 0006341X, 15410420. http://www.jstor.org/stable/20640622.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.sdlda")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
```

```
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.sgd`
Classification Stochastic Gradient Descent Learner

Description

Stochastic Gradient Descent for learning various linear models. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Initial parameter values

- `F`:
  - Has only 2 out of 5 original loss functions: 0 = hinge loss (SVM) and 1 = log loss (logistic regression) with 0 (hinge loss) still being the default
  - Reason for change: this learner should only contain loss functions appropriate for classification tasks

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.sgd")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
F	character	0	0, 1	-
L	numeric	0.01		$(-\infty, \infty)$
R	numeric	1e-04		$(-\infty, \infty)$
E	integer	500		$(-\infty, \infty)$
C	numeric	0.001		$(-\infty, \infty)$
N	logical	-	TRUE, FALSE	-
M	logical	-	TRUE, FALSE	-
S	integer	1		$(-\infty, \infty)$
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifSGD`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifSGD$new()`
- `LearnerClassifSGD$marshal()`
- `LearnerClassifSGD$unmarshal()`
- `LearnerClassifSGD$clone()`

`LearnerClassifSGD$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifSGD$new()
```

`LearnerClassifSGD$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifSGD$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifSGD$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifSGD$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifSGD$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifSGD$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.sgd")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.simple_logistic

Classification SimpleLogistic Learner

Description

LogitBoost with simple regression functions as base learners. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.simple_logistic")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
I	integer	-		$(-\infty, \infty)$
S	logical	FALSE	TRUE, FALSE	-
P	logical	FALSE	TRUE, FALSE	-
M	integer	-		$(-\infty, \infty)$
H	integer	50		$(-\infty, \infty)$
W	numeric	0		$(-\infty, \infty)$
A	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifSimpleLogistic`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifSimpleLogistic$new()`
- `LearnerClassifSimpleLogistic$marshal()`

- `LearnerClassifSimpleLogistic$unmarshal()`
- `LearnerClassifSimpleLogistic$clone()`

`LearnerClassifSimpleLogistic$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifSimpleLogistic$new()
```

`LearnerClassifSimpleLogistic$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifSimpleLogistic$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifSimpleLogistic$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifSimpleLogistic$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifSimpleLogistic$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifSimpleLogistic$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Landwehr, Niels, Hall, Mark, Frank, Eibe (2005). "Logistic model trees." *Machine learning*, **59**(1), 161–205.

Sumner M, Frank E, Hall M (2005). "Speeding up Logistic Model Tree Induction." In *9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 675-683.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.simple_logistic")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.smo`

Classification Support Vector Machine Learner

Description

Support Vector classifier trained with John Platt's sequential minimal optimization algorithm. Calls `RWeka::SMO()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:

    - original id: do-not-check-capabilities
- `num_decimal_places`:
    - original id: num-decimal-places
- `batch_size`:
    - original id: batch-size
- `E_poly`:
    - original id: E
- `L_poly`:
    - original id: L
- `C_poly`:
    - original id: C
- `C_logistic`:
    - original id: C
- `R_logistic`:
    - original id: R
- `M_logistic`:
    - original id: M
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.smo")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels
subset	untyped	-	
na.action	untyped	-	
no_checks	logical	FALSE	TRUE, FALSE
C	numeric	1	
N	character	0	0, 1, 2
L	numeric	0.001	

P	numeric	1e-12	
M	logical	FALSE	TRUE, FALSE
V	integer	-1	
W	integer	1	
K	character	PolyKernel	NormalizedPolyKernel, PolyKernel, Puk, RB
calibrator	untyped	"weka.classifiers.functions.Logistic"	
E_poly	numeric	1	
L_poly	logical	FALSE	TRUE, FALSE
C_poly	integer	25007	
C_logistic	logical	FALSE	TRUE, FALSE
R_logistic	numeric	-	
M_logistic	integer	-1	
output_debug_info	logical	FALSE	TRUE, FALSE
do_not_check_capabilities	logical	FALSE	TRUE, FALSE
num_decimal_places	integer	2	
batch_size	integer	100	
options	untyped	NULL	

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifSMO`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifSMO$new()`
- `LearnerClassifSMO$marshal()`
- `LearnerClassifSMO$unmarshal()`
- `LearnerClassifSMO$clone()`

`LearnerClassifSMO$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifSMO$new()
```

`LearnerClassifSMO$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifSMO$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifSMO$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifSMO$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifSMO$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifSMO$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Platt J (1998). "Fast Training of Support Vector Machines using Sequential Minimal Optimization." In Schoelkopf B, Burges C, Smola A (eds.), *Advances in Kernel Methods - Support Vector Learning*. MIT Press. https://www.microsoft.com/en-us/research/publication/sequential-minimal-optimization-a-fas

Keerthi S, Shevade S, Bhattacharyya C, Murthy K (2001). "Improvements to Platt's SMO Algorithm for SVM Classifier Design." *Neural Computation*, **13**(3), 637-649.

Hastie T, Tibshirani R (1998). "Classification by Pairwise Coupling." In Jordan MI, Kearns MJ, Solla SA (eds.), *Advances in Neural Information Processing Systems*, volume 10.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.smo")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.sparseLDA`

Sparse Discriminant Analysis

Description

Sparse Linear Discriminant Analysis for classification. Calls `sparseLDA::sda()` from **sparseLDA**.

Custom mlr3 parameters

- `Q` is set internally to `min(n_features, n_classes - 1)` when not supplied.
- `stop` is not exposed because it depends on the task.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.sparseLDA")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **sparseLDA**, **MASS**, **elasticnet**

Parameters

Id	Type	Default	Levels	Range
lambda	numeric	1e-06		$[0, \infty)$
maxIte	integer	100		$[0, \infty)$
tol	numeric	1e-06		$[0, \infty)$
trace	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifSparseLDA`

Methods

Public methods:

- `LearnerClassifSparseLDA$new()`
- `LearnerClassifSparseLDA$clone()`

`LearnerClassifSparseLDA$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifSparseLDA$new()
```

`LearnerClassifSparseLDA$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifSparseLDA$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Clemmensen, Line, Hastie, Trevor, Witten, Daniela, Ersboll, Bjarne (2011). "Sparse discriminant analysis." *Journal of the American Statistical Association*, **106**(496), 1519–1531. doi:10.1198/jasa.2011.tm09728.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.sparseLDA")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_classif.stepPlr

Classification Logistic Regression Learner

Description

Logistic regression with a quadratic penalization on the coefficient. Calls `stepPlr::plr()` from **stepPlr**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.stepPlr")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **stepPlr**

Parameters

Id	Type	Default	Levels	Range
cp	character	aic	aic, bic	-
lambda	numeric	1e-04		$[0, \infty)$
offset.coefficients	untyped	-		-
offset.subset	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifStepPlr`

Methods

Public methods:

- `LearnerClassifStepPlr$new()`
- `LearnerClassifStepPlr$clone()`

`LearnerClassifStepPlr$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifStepPlr$new()
```

`LearnerClassifStepPlr$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifStepPlr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Park, Young M, Hastie, Trevor (2007). "Penalized logistic regression for detecting gene interactions." *Biostatistics*, **9**(1), 30-50. ISSN 1465-4644. doi:10.1093/biostatistics/kxm010. https://doi.org/10.1093/biostatistics/kxm010.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.stepPlr")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_classif.tabfm`

TabFM Classification Learner

Description

Tabular foundation model developed by Google Research. Uses **reticulate** to interface with the `tabfm` Python package. The model makes zero-shot predictions via in-context learning and is not trained on the data in the usual sense. Features are passed to `tabfm` as a data frame, so that categorical features are ordinal encoded and missing values are imputed by the package's own preprocessing pipeline.

Installation

This learner relies on **reticulate** to handle Python dependencies. It is not necessary to install any Python package manually in advance or specify a Python environment via `reticulate::use_python()`, `reticulate::use_virtualenv()`, `reticulate::use_condaenv()`, or `reticulate::use_miniconda()`. By calling `$train()` or `$predict()`, the required Python packages will be installed automatically, if not already. The backend parameter selects which extra is installed: `"jax"` installs `tabfm[jax]` and `"pytorch"` installs `tabfm[pytorch]`. Reticulate will then configure and initialize an ephemeral environment satisfying those requirements, unless an existing environment (e.g., `"r-reticulate"`) in reticulate's Order of Discovery contains all the necessary packages.

You may also manually install `tabfm` into a Python environment following the official installation guide and specify the environment via `reticulate::use_*()` before calling `$train()` or `$predict()`.

On the first call to `$train()`, the pre-trained model weights (about 100 MB) are downloaded from Hugging Face and cached for subsequent calls.

Saving a Learner

In order to save a `LearnerClassifTabFM` for later usage, it is necessary to call the `$marshal()` method on the `Learner` before writing it to disk, as the object will otherwise not be saved correctly. After loading a marshaled `LearnerClassifTabFM` into R again, you then need to call `$unmarshal()` to transform it into a useable state. The several gigabyte pre-trained backbone is not serialized; it is detached before marshaling and reloaded from the Hugging Face cache on `$unmarshal()`.

Custom mlr3 parameters

- `backend` selects the computational backend, either `"jax"` or `"pytorch"`. It also determines which pre-trained weights are downloaded and which Python packages are installed.
- `device` is only used by the `"pytorch"` backend and is passed as a string to the model loader, e.g., `"cpu"` or `"cuda"`. The `"jax"` backend ignores this parameter and selects the device automatically.
- `norm_methods` accepts a single string or a character vector with values from `"none"`, `"power"`, `"quantile"`, `"quantile_rtdl"`, and `"robust"`.
- `n_feature_crosses` and `n_svd_features` accept either a non-negative integer or the string `"sqrt"`.
- `cache_context` is only supported by the `"pytorch"` backend.
- `random_state` accepts either an integer or the special value `"None"` which corresponds to `None` in Python.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.tabfm")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **reticulate**

Parameters

Id	Type	Default	Levels	Range
backend	character	-	jax, pytorch	-
device	untyped	-		-
n_estimators	integer	32		$[1, \infty)$
norm_methods	untyped	-		-
feat_shuffle_method	character	random	random, none	-
class_shift	logical	TRUE	TRUE, FALSE	-
permute_categorical	logical	FALSE	TRUE, FALSE	-
outlier_threshold	numeric	4		$(-\infty, \infty)$
max_num_features	integer	500		$[1, \infty)$
max_num_rows	integer	-		$[1, \infty)$
softmax_temperature	numeric	0.9		$[0, \infty)$
average_logits	logical	TRUE	TRUE, FALSE	-
use_amp	logical	TRUE	TRUE, FALSE	-
batch_size	integer	1		$[1, \infty)$
random_state	integer	42		$(-\infty, \infty)$
verbose	logical	FALSE	TRUE, FALSE	-
cat_encoder_mode	character	appearance	appearance, frequency	-
binary_calibration_method	character	-	platt	-
multiclass_calibration_method	character	-	vector	-
num_folds_for_cv	integer	5		$[2, \infty)$
n_feature_crosses	untyped	0L		-
n_svd_features	untyped	0L		-
total_svd_pool	integer	-		$[1, \infty)$
enable_nnls	logical	FALSE	TRUE, FALSE	-
nnls_beta	numeric	0.75		$(-\infty, \infty)$
calibration_lambda	numeric	0.01		$[0, \infty)$
min_rows_for_single_val_split	integer	2000		$[0, \infty)$
cache_context	logical	FALSE	TRUE, FALSE	-
maybe_quantize_kv_cache	logical	TRUE	TRUE, FALSE	-
keep_cache_on_device	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifTabFM`

Active bindings

`marshaled (logical(1))`
: Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifTabFM$new()`
- `LearnerClassifTabFM$marshal()`
- `LearnerClassifTabFM$unmarshal()`
- `LearnerClassifTabFM$clone()`

`LearnerClassifTabFM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifTabFM$new()
```

`LearnerClassifTabFM$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifTabFM$marshal(...)
```

Arguments:

`...` (any)
: Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifTabFM$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifTabFM$unmarshal(...)
```

Arguments:

`...` (any)
: Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifTabFM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifTabFM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Google Research (2026). "TabFM: Tabular Foundation Models." https://github.com/google-research/tabfm.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

`mlr_learners_classif.tabpfn`

TabPFN Classification Learner

Description

Foundation model for tabular data. Uses **reticulate** to interface with the `tabpfn` Python package.

Installation

This learner relies on **reticulate** to handle Python dependencies. It is not necessary to install any Python package manually in advance or specify a Python environment via `reticulate::use_python()`, `reticulate::use_virtualenv()`, `reticulate::use_condaenv()`, or `reticulate::use_miniconda()`. By calling `$train()` or `$predict()`, the required Python packages (`tabpfn`, `torch`, etc.) will be installed automatically, if not already. Reticulate will then configure and initialize an ephemeral environment satisfying those requirements, unless an existing environment (e.g., `"r-reticulate"`) in reticulate's Order of Discovery contains all the necessary packages.

You may also manually install `tabpfn` into a Python environment following the official installation guide and specify the environment via `reticulate::use_*()` before calling `$train()` or `$predict()`. Note that the GPU version of PyTorch cannot be loaded by `reticulate::use_condaenv()` from a conda environment. To use a conda environment, please install the CPU version of PyTorch.

License acceptance

Since `tabpfn` version 8, the default model version (v3) requires a one-time acceptance of the Prior Labs license before the model weights can be downloaded. On the first call to `$train()` or `$predict()`, `tabpfn` opens a browser window to log in and accept the license, and caches the resulting token for subsequent calls. In non-interactive environments (e.g., on a server or in continuous integration), this browser flow fails. Instead, log in at `https://platform.priorlabs.ai`, accept the license on the licenses tab, copy your API key from `https://platform.priorlabs.ai/account`, and set it as the environment variable `TABPFN_TOKEN` before training.

Saving a Learner

In order to save a `LearnerClassifTabPFN` for later usage, it is necessary to call the `$marshal()` method on the `Learner` before writing it to disk, as the object will otherwise not be saved correctly. After loading a marshaled `LearnerClassifTabPFN` into R again, you then need to call `$unmarshal()` to transform it into a useable state.

Initial parameter values

- `n_jobs` is initialized to 1 to avoid threading conflicts with **future**.

Custom mlr3 parameters

- `categorical_features_indices` uses R indexing instead of zero-based Python indexing. It is only needed to mark numeric or logical features as categorical, because `factor`, `ordered`, and `character` features are always encoded as categorical by `tabpfn`. The level order of `ordered` features is not preserved, they are treated like unordered `factor` features.
- `device` must be a string. If set to `"auto"`, the behavior is the same as original. Otherwise, the string is passed as argument to `torch.device()` to create a device.
- `inference_precision` must be `"auto"`, `"autocast"`, or a `torch.dtype` string, e.g., `"torch.float32"`, `"torch.float64"`, etc. Non-float dtypes are not supported.
- `inference_config` is currently not supported.
- `tuning_config` must be passed as a named list, which is converted to a Python dictionary. It enables post-hoc tuning towards `eval_metric` and accepts the keys `calibrate_temperature`, `tune_decision_thresholds`, `tuning_holdout_frac`, and `tuning_n_folds`.
- `eval_metric` only affects predictions when `tuning_config` is also set, in which case the softmax temperature and decision thresholds are tuned towards the chosen metric during training.
- `n_jobs` is deprecated upstream in favor of `n_preprocessing_jobs` and is only kept for backward compatibility.
- `random_state` accepts either an integer or the special value `"None"` which corresponds to `None` in Python. Following the original Python implementation, the default `random_state` is `0`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.tabpfn")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **reticulate**

Parameters

Id	Type	Default	Levels
n_estimators	integer	8	
auto_scale_n_estimators	logical	TRUE	TRUE, FALSE
categorical_features_indices	untyped	-	
softmax_temperature	numeric	0.9	
balance_probabilities	logical	FALSE	TRUE, FALSE
average_before_softmax	logical	FALSE	TRUE, FALSE
model_path	untyped	"auto"	
device	untyped	"auto"	
ignore_pretraining_limits	logical	FALSE	TRUE, FALSE
inference_precision	character	auto	auto, autocast, torch.float32, torch.float, torch.float64, torch.do
fit_mode	character	fit_preprocessors	low_memory, fit_preprocessors, fit_with_cache, batched
memory_saving_mode	untyped	"auto"	
keep_cache_on_device	logical	TRUE	TRUE, FALSE
random_state	integer	0	
n_jobs	integer	-	
n_preprocessing_jobs	integer	1	
differentiable_input	logical	FALSE	TRUE, FALSE
eval_metric	character	-	accuracy, balanced_accuracy, roc_auc, f1, log_loss
tuning_config	untyped	-	
show_progress_bar	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifTabPFN`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifTabPFN$new()`
- `LearnerClassifTabPFN$marshal()`
- `LearnerClassifTabPFN$unmarshal()`
- `LearnerClassifTabPFN$clone()`

`LearnerClassifTabPFN$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifTabPFN$new()
```

`LearnerClassifTabPFN$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifTabPFN$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifTabPFN$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifTabPFN$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifTabPFN$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifTabPFN$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

b-zhou

References

Hollmann, Noah, Müller, Samuel, Purucker, Lennart, Krishnakumar, Arjun, Körfer, Max, Hoo, Bin S, Schirrmeister, Tibor R, Hutter, Frank (2025). "Accurate predictions on small data with a tabular foundation model." *Nature*. doi:10.1038/s41586024083286. https://www.nature.com/articles/s41586-024-08328-6.

Hollmann, Noah, Müller, Samuel, Eggensperger, Katharina, Hutter, Frank (2023). "TabPFN: A transformer that solves small tabular classification problems in a second." In *International Conference on Learning Representations 2023*.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

`mlr_learners_classif.voted_perceptron`
Classification Voted Perceptron Learner

Description

Voted Perceptron Algorithm by Freund and Schapire. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("classif.voted_perceptron")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
I	integer	1		$[1, \infty)$
E	numeric	1		$(-\infty, \infty)$
S	integer	1		$(-\infty, \infty)$

M	integer	10000		$(-\infty, \infty)$
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifVotedPerceptron`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerClassifVotedPerceptron$new()`
- `LearnerClassifVotedPerceptron$marshal()`
- `LearnerClassifVotedPerceptron$unmarshal()`
- `LearnerClassifVotedPerceptron$clone()`

`LearnerClassifVotedPerceptron$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerClassifVotedPerceptron$new()
```

`LearnerClassifVotedPerceptron$marshal()`: Marshal the learner's model.

Usage:

```
LearnerClassifVotedPerceptron$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerClassifVotedPerceptron$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerClassifVotedPerceptron$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerClassifVotedPerceptron$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerClassifVotedPerceptron$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Freund Y, Schapire RE (1998). "Large margin classification using the perceptron algorithm." In *11th Annual Conference on Computational Learning Theory*, 209-217.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("classif.voted_perceptron")
print(learner)

# Define a Task
task = tsk("sonar")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
```

```
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_cmprsk.coxboost`

Competing Risks Cox Learner with Likelihood Based Boosting

Description

Fits a competing risks Cox model using likelihood based boosting. Calls `CoxBoost::CoxBoost()` or `CoxBoost::cv.CoxBoost()` from package **CoxBoost**. Predicts cumulative incidence functions (one CIF matrix per cause, rows = observations, columns = times). If `times` are not provided for prediction, the unique event times (across causes) from the training set are used by default.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("cmprsk.coxboost")
```

Meta Information

- Task type: "cmprsk"
- Predict Types: "cif"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3cmprsk**, **mlr3extralearners**, **CoxBoost**

Parameters

Id	Type	Default	Levels	Range
cv	logical	-	TRUE, FALSE	-
maxstepno	integer	100		$[0, \infty)$
K	integer	10		$[2, \infty)$
type	character	verweij	verweij, naive	-
folds	untyped	NULL		-
minstepno	integer	50		$[0, \infty)$
start.penalty	numeric	-		$(-\infty, \infty)$
iter.max	integer	10		$[1, \infty)$
upper.margin	numeric	0.05		$[0, 1]$
unpen.index	untyped	-		-
standardize	logical	TRUE	TRUE, FALSE	-
stepno	integer	100		$[0, \infty)$
penalty	numeric	-		$(-\infty, \infty)$

criterion	character	pscore	pscore, score, hpscore, hscore	-
stepsize.factor	numeric	1		$(-\infty, \infty)$
sf.scheme	character	sigmoid	sigmoid, linear	-
pendistmat	untyped	-		-
connected.index	untyped	-		-
x.is.01	logical	FALSE	TRUE, FALSE	-
return.score	logical	TRUE	TRUE, FALSE	-
trace	logical	FALSE	TRUE, FALSE	-
cmprsk	character	-	csh, ccsh	-
coupled.strata	logical	TRUE	TRUE, FALSE	-
at.step	untyped	-		-
times	untyped	-		-

Custom mlr3 parameters

- `cv`: If `TRUE`, uses cross-validation with `CoxBoost::cv.CoxBoost()` to select the optimal number of boosting steps, then refits the final model using `CoxBoost::CoxBoost()` with this value. Initialized to `FALSE` by default.
- `penalty`: If set to `"optimCoxBoostPenalty"`, uses `CoxBoost::optimCoxBoostPenalty()` to determine the optimal penalty value and number of steps (via `CoxBoost::cv.CoxBoost()`) for the final model.

Initial parameter values

- `cmprsk`: Type of competing risks analysis. Initialized to `"csh"` (**cause-specific hazards**) which models each cause separately, treating other causes as censored. `"ccsh"` (**combined cause-specific hazards**) is another option which combines information across causes for variable selection using a joint criterion.

Super classes

`mlr3::Learner` -> `mlr3cmprsk::LearnerCompRisks` -> `LearnerCompRisksCoxboost`

Methods

Public methods:

- `LearnerCompRisksCoxboost$new()`
- `LearnerCompRisksCoxboost$selected_features()`
- `LearnerCompRisksCoxboost$clone()`

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

Usage:

```
LearnerCompRisksCoxboost$new()
```

`LearnerCompRisksCoxboost$selected_features()`: Returns the set of cause-specific selected features which have non-zero coefficients. Calls the internal `coef.CoxBoost()` function.

Usage:

```
LearnerCompRisksCoxboost$selected_features(at_step = NULL, cause = 1)
```

Arguments:

`at_step (integer(1))`
Which boosting step to get the coefficients for. If no step is given (default), the final boosting step is used.

`cause (integer(1))`
Integer value indicating the event of interest.

Returns: (`character()`) vector of feature names.

`LearnerCompRisksCoxboost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerCompRisksCoxboost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bblodfon

References

Binder, Harald, Allignol, Arthur, Schumacher, Martin, Beyersmann, Jan (2009). "Boosting for high-dimensional time-to-event data with competing risks." *Bioinformatics*, **25**(7), 890–896.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("cmprsk.coxboost", stepno = 10)
print(learner)

# Define a Task
task = tsk("pbc")
task$col_roles$feature = c("age", "chol", "copper", "platelet", "trig",
 "albumin", "alk.phos", "ast", "bili", "protime")

# Stratification based on event
task$set_col_roles(cols = "status", add_to = "stratum")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$selected_features(cause = 1)) # for cause = 1
print(learner$selected_features(cause = 2)) # for cause = 2

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_cmprsk.rfsrc

Random Forest Competing Risks Learner

Description

Random survival forests for competing risks. Calls `randomForestSRC::rfsrc()` from **randomForestSRC**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("cmprsk.rfsrc")
```

Meta Information

- Task type: “cmprsk”
- Predict Types: “cif”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3cmprsk**, **mlr3extralearners**, **randomForestSRC**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
mtry	integer	-		$[1, \infty)$
mtry.ratio	numeric	-		$[0, 1]$
nodesize	integer	15		$[1, \infty)$
nodedepth	integer	-		$[1, \infty)$
splitrule	character	logrankCR	logrankCR, logrank	-
nsplit	integer	10		$[0, \infty)$
importance	character	FALSE	FALSE, TRUE, none, anti, permute, random	-
block.size	integer	10		$[1, \infty)$
bootstrap	character	by.root	by.root, by.node, none, by.user	-
samptype	character	swor	swor, swr	-
samp	untyped	-		-
membership	logical	FALSE	TRUE, FALSE	-
sampsize	untyped	-		-
sampsize.ratio	numeric	-		$[0, 1]$
na.action	character	na.omit	na.omit, na.impute	-
nimpute	integer	1		$[1, \infty)$
ntime	integer	150		$[0, \infty)$
cause	untyped	-		-
proximity	character	FALSE	FALSE, TRUE, inbag, oob, all	-
distance	character	FALSE	FALSE, TRUE, inbag, oob, all	-
forest.wt	character	FALSE	FALSE, TRUE, inbag, oob, all	-
xvar.wt	untyped	-		-
split.wt	untyped	-		-
forest	logical	TRUE	TRUE, FALSE	-
var.used	character	FALSE	FALSE, all.trees	-
split.depth	character	FALSE	FALSE, all.trees, by.tree	-
seed	integer	-		$(-\infty, -1]$
do.trace	logical	FALSE	TRUE, FALSE	-
get.tree	untyped	-		-
outcome	character	train	train, test	-
ptn.count	integer	0		$[0, \infty)$
cores	integer	1		$[1, \infty)$
save.memory	logical	FALSE	TRUE, FALSE	-
use.uno	logical	TRUE	TRUE, FALSE	-
perf.type	character	-	none	-
case.depth	logical	FALSE	TRUE, FALSE	-
marginal.xvar	untyped	NULL		-

Initial parameter values

- `ntime`: Number of time points to coerce the observed event times for use in the estimated cumulative incidence functions during prediction. We changed the default value of `150` to `0`, meaning we now use all the **unique event times from the train set** across all competing causes.

Custom mlr3 parameters

- `mtry`: This hyperparameter can alternatively be set via the added hyperparameter `mtry.ratio` as `mtry = max(ceiling(mtry.ratio * n_features), 1)`. Note that `mtry` and `mtry.ratio` are mutually exclusive.
- `sampsize`: This hyperparameter can alternatively be set via the added hyperparameter `sampsize.ratio` as `sampsize = max(ceiling(sampsize.ratio * n_obs), 1)`. Note that `sampsize` and `sampsize.ratio` are mutually exclusive.
- `cores`: This value is set as the option `rf.cores` during training and is set to 1 by default.

Super classes

`mlr3::Learner` -> `mlr3cmprsk::LearnerCompRisks` -> `LearnerCompRisksRandomForestSRC`

Methods

Public methods:

- `LearnerCompRisksRandomForestSRC$new()`
- `LearnerCompRisksRandomForestSRC$importance()`
- `LearnerCompRisksRandomForestSRC$selected_features()`
- `LearnerCompRisksRandomForestSRC$oob_error()`
- `LearnerCompRisksRandomForestSRC$clone()`

`LearnerCompRisksRandomForestSRC$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerCompRisksRandomForestSRC$new()
```

`LearnerCompRisksRandomForestSRC$importance()`: The importance scores are extracted from the model slot `importance` and are cause-specific.

Usage:

```
LearnerCompRisksRandomForestSRC$importance(cause = 1)
```

Arguments:

`cause` Integer value indicating the event of interest

Returns: Named `numeric()`.

`LearnerCompRisksRandomForestSRC$selected_features()`: Selected features are extracted from the model slot `var.used`.

Note: Due to a known issue in `randomForestSRC`, enabling `var.used = "all.trees"` causes prediction to fail. Therefore, this setting should be used exclusively for feature selection purposes and not when prediction is required.

Usage:

```
LearnerCompRisksRandomForestSRC$selected_features()
```

Returns: `character()`.

`LearnerCompRisksRandomForestSRC$oob_error()`: Extracts the out-of-bag (OOB) cumulative incidence function (CIF) error from the model's `err.rate` slot.

If `cause = "mean"` (default), the function returns a weighted average of the cause-specific OOB errors, where the weights correspond to the observed proportion of events for each cause in the training data.

Usage:

```
LearnerCompRisksRandomForestSRC$oob_error(cause = "mean")
```

Arguments:

`cause` Integer (event type) or `"mean"` (default). Use a specific event type to retrieve its OOB error, or `"mean"` to compute the weighted average across causes.

Returns: `numeric()`.

`LearnerCompRisksRandomForestSRC$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerCompRisksRandomForestSRC$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bblodfon

References

Ishwaran, H., Gerds, A. T, Kogalur, B. U, Moore, D. R, Gange, J. S, Lau, M. B (2014). "Random survival forests for competing risks." *Biostatistics*, **15**(4), 757–773. doi:10.1093/BIOSTATISTICS/KXU010. https://doi.org/10.1093/BIOSTATISTICS/KXU010.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners

- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("cmprsk.rfsrc", importance = "TRUE")
print(learner)

# Define a Task
task = tsk("pbc")

# Stratification based on event
task$set_col_roles(cols = "status", add_to = "stratum")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance(cause = 1)) # VIMP for cause = 1
print(learner$importance(cause = 2)) # VIMP for cause = 2
print(learner$oob_error()) # weighted-mean across causes

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.abess

Regression Abess Learner

Description

Adaptive best-subset selection for regression. Calls `abess::abess()` from **abess**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.abess")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “integer”, “numeric”
- Required Packages: **mlr3**, **abess**

Parameters

Id	Type	Default	Levels	Range
family	character	gaussian	gaussian, mgaussian, poisson, gamma	-
tune.path	character	sequence	sequence, gsection	-
tune.type	character	gic	gic, aic, bic, ebic, cv	-
normalize	integer	NULL		$(-\infty, \infty)$
support.size	untyped	NULL		-
c.max	integer	2		$[1, \infty)$
gs.range	untyped	NULL		-
lambda	numeric	0		$[0, \infty)$
always.include	untyped	NULL		-
group.index	untyped	NULL		-
init.active.set	untyped	NULL		-
splicing.type	integer	2		$[1, 2]$
max.splicing.iter	integer	20		$[1, \infty)$
screening.num	integer	NULL		$[0, \infty)$
important.search	integer	NULL		$[0, \infty)$
warm.start	logical	TRUE	TRUE, FALSE	-
nfolds	integer	5		$(-\infty, \infty)$
foldid	untyped	NULL		-
cov.update	logical	FALSE	TRUE, FALSE	-
newton	character	exact	exact, approx	-
newton.thresh	numeric	1e-06		$[0, \infty)$
max.newton.iter	integer	NULL		$[1, \infty)$
early.stop	logical	FALSE	TRUE, FALSE	-
ic.scale	numeric	1		$[0, \infty)$
num.threads	integer	0		$[0, \infty)$
seed	integer	1		$(-\infty, \infty)$

Initial parameter values

- `num.threads`: This parameter is initialized to 1 (default is 0) to avoid conflicts with the mlr3 parallelization.
- `family`: Depends on the task type, if the parameter `family` is `NULL`, it is set to `"binomial"` for binary classification tasks and to `"multinomial"` for multiclass classification problems.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrAbess`

Methods

Public methods:

- `LearnerRegrAbess$new()`
- `LearnerRegrAbess$selected_features()`
- `LearnerRegrAbess$clone()`

`LearnerRegrAbess$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrAbess$new()
```

`LearnerRegrAbess$selected_features()`: Extract the name of selected features from the model by `abess::extract()`.

Usage:

```
LearnerRegrAbess$selected_features()
```

Returns: The names of selected features

`LearnerRegrAbess$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrAbess$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bbayukari

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.abess")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.aorsf

Oblique Random Forest Regressor

Description

Accelerated oblique random regression forest. Calls `aorsf::orsf()` from **aorsf**. Note that although the learner has the property `"missing"` and it can in principle deal with missing values, the behavior has to be configured using the parameter `na_action`.

Initial parameter values

- `n_thread`: This parameter is initialized to 1 (default is 0) to avoid conflicts with the mlr3 parallelization.
- `pred_simplify` has to be TRUE, otherwise response is NA in prediction

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.aorsf")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3extralearners**, **aorsf**

Parameters

Id	Type	Default	Levels	Range
attach_data	logical	TRUE	TRUE, FALSE	-
epsilon	numeric	1e-09		$[0, \infty)$
importance	character	anova	none, anova, negate, permute	-
importance_max_pvalue	numeric	0.01		$[1e-04, 0.9999]$
leaf_min_events	integer	1		$[1, \infty)$
leaf_min_obs	integer	5		$[1, \infty)$
max_iter	integer	20		$[1, \infty)$
method	character	glm	glm, net, pca, random	-
mtry	integer	NULL		$[1, \infty)$
mtry_ratio	numeric	-		$[0, 1]$
n_retry	integer	3		$[0, \infty)$
n_split	integer	5		$[1, \infty)$
n_thread	integer	-		$[0, \infty)$
n_tree	integer	500		$[1, \infty)$
na_action	character	fail	fail, impute_meanmode	-
net_mix	numeric	0.5		$(-\infty, \infty)$
oobag	logical	FALSE	TRUE, FALSE	-
oobag_eval_every	integer	NULL		$[1, \infty)$
oobag_fun	untyped	NULL		-
oobag_pred_type	character	mean	none, leaf, mean	-
pred_aggregate	logical	TRUE	TRUE, FALSE	-
sample_fraction	numeric	0.632		$[0, 1]$
sample_with_replacement	logical	TRUE	TRUE, FALSE	-
scale_x	logical	FALSE	TRUE, FALSE	-
split_min_events	integer	5		$[1, \infty)$
split_min_obs	integer	10		$[1, \infty)$
split_min_stat	numeric	NULL		$[0, \infty)$
split_rule	character	variance	variance	-
target_df	integer	NULL		$[1, \infty)$
tree_seeds	integer	NULL		$[1, \infty)$
verbose_progress	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrObliqueRandomForest`

Methods

Public methods:

- `LearnerRegrObliqueRandomForest$new()`
- `LearnerRegrObliqueRandomForest$oob_error()`
- `LearnerRegrObliqueRandomForest$importance()`
- `LearnerRegrObliqueRandomForest$clone()`

`LearnerRegrObliqueRandomForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrObliqueRandomForest$new()
```

`LearnerRegrObliqueRandomForest$oob_error()`: OOB concordance error extracted from the model slot `eval_oobag$stat_values`

Usage:

```
LearnerRegrObliqueRandomForest$oob_error()
```

Returns: `numeric()`.

`LearnerRegrObliqueRandomForest$importance()`: The importance scores are extracted from the model.

Usage:

```
LearnerRegrObliqueRandomForest$importance()
```

Returns: Named `numeric()`.

`LearnerRegrObliqueRandomForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrObliqueRandomForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.aorsf")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.bam *Regression Fast Generalized Additive Model Learner*

Description

Fast generalized additive models for large datasets. Calls `mgcv::bam()` from package **mgcv**. For smaller datasets, use `regr.gam` instead.

Formula

A gam formula specific to the task at hand is required for the `formula` parameter (see example and `?mgcv::formula.gam`). Beware, if no formula is provided, a fallback formula is used that will make the model behave like a glm (this behavior is required for the unit tests). Only features specified in the formula will be used, superseding columns with col_roles "feature" in the task.

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated during training via the `offset` argument in `mgcv::bam()`. No offset is applied during prediction for this learner.

Custom mlr3 parameters

- `control` is passed to `mgcv::gam.control()` as a list. Unlike `regr.gam`, the individual control options are not exposed as hyperparameters, because `nthreads` is both a control option and an argument of `mgcv::bam()`.

- `discrete` is a parameter of both `mgcv::bam()` and `mgcv::predict.bam()`, with different defaults. Only the training parameter is exposed, and prediction uses the default of `mgcv::predict.bam()`.
- Only the training parameter `nthreads` is tagged with `"threads"`, so `mlr3::set_threads()` does not change the prediction parameter `n.threads`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.bam")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **mgcv**

Parameters

Id	Type	Default	Levels	Range
formula	untyped	-		-
family	character	gaussian	gaussian, poisson	-
method	character	fREML	fREML, GCV.Cp, GACV.Cp, REML, P-REML, ML, P-ML	-
control	untyped	list()		-
select	logical	FALSE	TRUE, FALSE	-
scale	numeric	0		$(-\infty, \infty)$
gamma	numeric	1		$[1, \infty)$
knots	untyped	NULL		-
sp	untyped	NULL		-
min.sp	untyped	NULL		-
paraPen	untyped	NULL		-
chunk.size	integer	10000		$[1, \infty)$
rho	numeric	0		$(-\infty, \infty)$
AR.start	untyped	NULL		-
discrete	logical	FALSE	TRUE, FALSE	-
nthreads	integer	1		$[1, \infty)$
use.chol	logical	FALSE	TRUE, FALSE	-
samfrac	numeric	1		$[0, 1]$
coef	untyped	NULL		-
drop.unused.levels	logical	TRUE	TRUE, FALSE	-
G	untyped	NULL		-
drop.intercept	logical	NULL	TRUE, FALSE	-
in.out	untyped	NULL		-
nei	untyped	NULL		-
cluster	untyped	NULL		-
gc.level	integer	0		$[0, \infty)$

terms	untyped	NULL	-
exclude	untyped	NULL	-
block.size	integer	50000	$(-\infty, \infty)$
n.threads	integer	1	$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBam`

Methods

Public methods:

- `LearnerRegrBam$new()`
- `LearnerRegrBam$clone()`

`LearnerRegrBam$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBam$new()
```

`LearnerRegrBam$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBam$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Tino-Rg

References

Hastie, J T, Tibshirani, J R (2017). *Generalized additive models*. Routledge.

Wood, Simon (2012). "mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML smoothness estimation."

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.

- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# simple example
t = tsk("mtcars")
l = lrn("regr.bam")
l$param_set$values$formula = mpg ~ cyl + am + s(disp) + s(hp)
l$train(t)
l$model
```

`mlr_learners_regr.bart`

Regression BART (Bayesian Additive Regression Trees) Learner

Description

Bayesian Additive Regression Trees are similar to gradient boosting algorithms. Calls `dbarts::bart()` from **dbarts**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.bart")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **dbarts**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	200		$[1, \infty)$
sigest	untyped	NULL		-
sigdf	numeric	3		$[1, \infty)$
sigquant	numeric	0.9		$[0, 1]$
k	numeric	2		$[0, \infty)$
power	numeric	2		$[0, \infty)$

base	numeric	0.95		$[0, 1]$
ndpost	integer	1000		$[1, \infty)$
nskip	integer	100		$[0, \infty)$
printevery	integer	100		$[0, \infty)$
keepevery	integer	1		$[1, \infty)$
keeptrainfits	logical	TRUE	TRUE, FALSE	-
usequants	logical	FALSE	TRUE, FALSE	-
numcut	integer	100		$[1, \infty)$
printcutoffs	integer	0		$(-\infty, \infty)$
verbose	logical	FALSE	TRUE, FALSE	-
nthread	integer	1		$(-\infty, \infty)$
keeptrees	logical	FALSE	TRUE, FALSE	-
keepcall	logical	TRUE	TRUE, FALSE	-
sampleronly	logical	FALSE	TRUE, FALSE	-
seed	integer	NA		$(-\infty, \infty)$
proposalprobs	untyped	NULL		-
splitprobs	untyped	NULL		-
keepsampler	logical	-	TRUE, FALSE	-
n.threads	integer	-		$(-\infty, \infty)$

Custom mlr3 parameters

- Parameter: `offset`
  - The parameter is removed, because only `dbarts::bart2` allows an offset during training, and therefore the offset parameter in `dbarts:::predict.bart` is irrelevant for `dbarts::dbart`.
- Parameter: `nchain`, `combineChains`, `combinechains`
  - The parameters are removed as parallelization of multiple models is handled by future.

Initial parameter values

- `keeptrees` is initialized to `TRUE` because it is required for prediction.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBart`

Methods

Public methods:

- `LearnerRegrBart$new()`
- `LearnerRegrBart$clone()`

`LearnerRegrBart$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBart$new()
```

`LearnerRegrBart$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBart$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ck37

References

Sparapani, Rodney, Spanbauer, Charles, McCulloch, Robert (2021). "Nonparametric machine learning and efficient computation with bayesian additive regression trees: the BART R package." *Journal of Statistical Software*, **97**, 1–66.

Chipman, A H, George, I E, McCulloch, E R (2010). "BART: Bayesian additive regression trees." *The Annals of Applied Statistics*, **4**(1), 266–298.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.bart")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.bcart`

Regression Bayesian CART Learner

Description

Bayesian CART regression model. Calls `tgp::bcart()` from **tgp**. For the predicted mean ZZ.km and for the predicted variance ZZ.ks2 are chosen.

Factor features are one-hot encoded with reference encoding before fitting, matching the behavior of the original mlr learner. If factors are present, `basemax` is set to the number of non-factor features so that tree proposals account for the numeric part of the design.

Initial parameter values

- `verb` is initialized to `0` to silence printing.
- `pred.n` is initialized to `FALSE` to skip prediction during training.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.bcart")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “se”
- Feature Types: “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3extralearners**, **tgp**

Parameters

Id	Type	Default	Levels	Range
bprior	character	bflat	b0, b0not, bflat, bmle, bmznot, bmzt	-
Ds2x	logical	FALSE	TRUE, FALSE	-
improv	logical	FALSE	TRUE, FALSE	-
itemps	untyped	NULL		-
krige	logical	TRUE	TRUE, FALSE	-
m0r1	logical	TRUE	TRUE, FALSE	-
pred.n	logical	-	TRUE, FALSE	-
R	integer	1		$[1, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
verb	integer	-		$[0, 4]$
zcov	logical	FALSE	TRUE, FALSE	-
tree	untyped	c(0.5, 2)		-
BTE	untyped	c(2000L, 7000L, 2L)		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBcart`

Methods

Public methods:

- `LearnerRegrBcart$new()`
- `LearnerRegrBcart$clone()`

`LearnerRegrBcart$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBcart$new()
```

`LearnerRegrBcart$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBcart$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Gramacy RB (2007). "tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models." *Journal of Statistical Software*, **19**(9), 1–46. doi:10.18637/jss.v019.i09.

Gramacy RB, Taddy M (2010). "Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with tgp Version 2, an R Package for Treed Gaussian Process Models." *Journal of Statistical Software*, **33**(6), 1–48. doi:10.18637/jss.v033.i06.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.bcart")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.bgp` *Regression Bayesian Gaussian Process Learner*

Description

Bayesian Gaussian process regression. Calls `tgp::bgp()` from **tgp**. For the predicted mean ZZ.km and for the predicted variance ZZ.ks2 are chosen.

Initial parameter values

- `verb` is initialized to `0` to silence printing.
- `pred.n` is initialized to `FALSE` to skip prediction during training.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.bgp")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **tgp**

Parameters

Id	Type	Default	Levels	Range
meanfn	character	linear	constant, linear	-
bprior	character	bflat	b0, b0not, bflat, bmle, bmznot, bmzt	-
corr	character	expsep	exp, expsep, matern, sim	-
BTE	untyped	c(1000L, 4000L, 2L)		-
R	integer	1		$[1, \infty)$
m0r1	logical	TRUE	TRUE, FALSE	-
itemps	untyped	NULL		-
pred.n	logical	-	TRUE, FALSE	-
krige	logical	TRUE	TRUE, FALSE	-
zcov	logical	FALSE	TRUE, FALSE	-
Ds2x	logical	FALSE	TRUE, FALSE	-
improv	logical	FALSE	TRUE, FALSE	-
nu	numeric	1.5		$(-\infty, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
verb	integer	-		$[0, 4]$
MAP	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBgp`

Methods

Public methods:

- `LearnerRegrBgp$new()`
- `LearnerRegrBgp$clone()`

`LearnerRegrBgp$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBgp$new()
```

`LearnerRegrBgp$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBgp$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Gramacy RB (2007). "tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models." *Journal of Statistical Software*, **19**(9), 1–46. doi:10.18637/jss.v019.i09.

Gramacy RB, Taddy M (2010). "Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with tgp Version 2, an R Package for Treed Gaussian Process Models." *Journal of Statistical Software*, **33**(6), 1–48. doi:10.18637/jss.v033.i06.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.

- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.bgp")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.bgpllm`

Regression Bayesian Gaussian Process LLM Learner

Description

Bayesian Gaussian process regression with jumps to the limiting linear model. Calls `tgp::bgpllm()` from **tgp**. For the predicted mean ZZ.km and for the predicted variance ZZ.ks2 are chosen.

Initial parameter values

- `verb` is initialized to `0` to silence printing.
- `pred.n` is initialized to `FALSE` to skip prediction during training.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.bgpllm")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “se”
- Feature Types: “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3extralearners**, **tgp**

Parameters

Id	Type	Default	Levels	Range
meanfn	character	linear	constant, linear	-
bprior	character	bflat	b0, b0not, bflat, bmle, bmznot, bmzt	-
corr	character	expsep	exp, expsep, matern, sim	-
gamma	untyped	c(10, 0.2, 0.7)		-
BTE	untyped	c(2000L, 4000L, 2L)		-
R	integer	1		$[1, \infty)$
m0r1	logical	TRUE	TRUE, FALSE	-
itemps	untyped	NULL		-
pred.n	logical	-	TRUE, FALSE	-
krige	logical	TRUE	TRUE, FALSE	-
zcov	logical	FALSE	TRUE, FALSE	-
Ds2x	logical	FALSE	TRUE, FALSE	-
improv	logical	FALSE	TRUE, FALSE	-
nu	numeric	1.5		$(-\infty, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
verb	integer	-		$[0, 4]$
MAP	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBgpllm`

Methods

Public methods:

- `LearnerRegrBgpllm$new()`
- `LearnerRegrBgpllm$clone()`

`LearnerRegrBgpllm$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBgpllm$new()
```

`LearnerRegrBgpllm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBgpllm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Gramacy RB (2007). "tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models." *Journal of Statistical Software*, **19**(9), 1–46. doi:10.18637/jss.v019.i09.

Gramacy RB, Taddy M (2010). "Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with tgp Version 2, an R Package for Treed Gaussian Process Models." *Journal of Statistical Software*, **33**(6), 1–48. doi:10.18637/jss.v033.i06.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.bgpllm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.blm` *Regression Bayesian Linear Model Learner*

Description

Bayesian linear regression model. Calls `tgp::blm()` from **tgp**. For the predicted mean ZZ.km and for the predicted variance ZZ.ks2 are chosen.

Initial parameter values

- `verb` is initialized to `0` to silence printing.
- `pred.n` is initialized to `FALSE` to skip prediction during training.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.blm")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “se”
- Feature Types: “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3extralearners**, **tgp**

Parameters

Id	Type	Default	Levels	Range
bprior	character	bflat	b0, b0not, bflat, bmle, bmznot, bmzt	-
Ds2x	logical	FALSE	TRUE, FALSE	-
improv	logical	FALSE	TRUE, FALSE	-
itemps	untyped	NULL		-
krige	logical	TRUE	TRUE, FALSE	-
m0r1	logical	TRUE	TRUE, FALSE	-
MAP	logical	TRUE	TRUE, FALSE	-
meanfn	character	linear	constant, linear	-
pred.n	logical	-	TRUE, FALSE	-
R	integer	1		$[1, \infty)$

trace	logical	FALSE	TRUE, FALSE	-
verb	integer	-		$[0, 4]$
zcov	logical	FALSE	TRUE, FALSE	-
BTE	untyped	c(1000L, 4000L, 3L)		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBlm`

Methods

Public methods:

- `LearnerRegrBlm$new()`
- `LearnerRegrBlm$clone()`

`LearnerRegrBlm$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBlm$new()
```

`LearnerRegrBlm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBlm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Gramacy RB (2007). "tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models." *Journal of Statistical Software*, **19**(9), 1–46. doi:10.18637/jss.v019.i09.

Gramacy RB, Taddy M (2010). "Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with tgp Version 2, an R Package for Treed Gaussian Process Models." *Journal of Statistical Software*, **33**(6), 1–48. doi:10.18637/jss.v033.i06.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.blm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.blockforest`

BlockForest Regression Learner

Description

Random forests for blocks of clinical and omics covariate data. Calls `blockForest::blockfor()` from package **blockForest**. The training model includes only the `$forest` slot, excluding the `paramvalues` and the `biased_oob_error_donotuse`.

In this learner, only the trained forest object (`$forest`) is retained. The optimized block-specific tuning parameters (`paramvalues`) and the biased OOB error estimate (`biased_oob_error_donotuse`) are discarded, as they are either not needed for downstream use or not reliable for performance estimation.

Initial parameter values

- `num.threads` is initialized to 1 to avoid conflicts with parallelization via **future**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.blockforest")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **blockForest**

Parameters

Id	Type	Default	Levels	Range
blocks	untyped	-		-
block.method	character	BlockForest	BlockForest, RandomBlock, BlockVarSel, VarProb, SplitWeights	-
num.trees	integer	2000		$[1, \infty)$
mtry	untyped	NULL		-
nsets	integer	300		$[1, \infty)$
num.trees.pre	integer	1500		$[1, \infty)$
splitrule	character	extratrees	extratrees, variance, maxstat	-
always.select.block	integer	0		$[0, 1]$
importance	character	-	none, impurity, impurity_corrected, permutation	-
num.threads	integer	-		$[1, \infty)$
seed	integer	NULL		$(-\infty, \propto$
verbose	logical	TRUE	TRUE, FALSE	-
se.method	character	infjack	jack, infjack	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBlockForest`

Methods

Public methods:

- `LearnerRegrBlockForest$new()`
- `LearnerRegrBlockForest$importance()`
- `LearnerRegrBlockForest$clone()`

`LearnerRegrBlockForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBlockForest$new()
```

`LearnerRegrBlockForest$importance()`: The importance scores are extracted from the model slot `variable.importance`.

Usage:

```
LearnerRegrBlockForest$importance()
```

Returns: Named `numeric()`.

`LearnerRegrBlockForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBlockForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bblodfon

References

Hornung, R., Wright, N. M (2019). "Block Forests: Random forests for blocks of clinical and omics covariate data." *BMC Bioinformatics*, **20**(1), 1–17. doi:10.1186/s128590192942y. https://doi.org/10.1186/s12859-019-2942-y.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# check task's features
task$feature_names

# partition features to 2 blocks
blocks = list(bl1 = 1:3, bl2 = 4:10)

# define learner
learner = lrn("regr.blockforest", blocks = blocks,
              importance = "permutation", nsets = 10,
              num.trees = 50, num.trees.pre = 10, splitrule = "variance")

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# feature importance
learner$importance()

# Make predictions for the test observations
pred = learner$predict(task, row_ids = ids$test)
pred

# Score the predictions
pred$score()
```

`mlr_learners_regr.botorch_fullybayesian`

BoTorch Fully Bayesian SingleTaskGP Regression Learner

Description

Fully Bayesian Gaussian Process via botorch, using the `SaasFullyBayesianSingleTaskGP` model. Unlike standard GP models that use MAP estimation for hyperparameters, this model uses MCMC (NUTS) to learn full posterior distributions over kernel hyperparameters. See here for more details. Uses **reticulate** to interface with Python.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.botorch_fullybayesian")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **reticulate**

Parameters

Id	Type	Default	Levels	Range
device	character	cpu	cpu, cuda	-
input_transform	character	-	normalize, standardize, log10, warp, none	-
outcome_transform	character	-	standardize, none	-
warmup_steps	integer	-		$[1, \infty)$
num_samples	integer	-		$[1, \infty)$
thinning	integer	-		$[1, \infty)$
max_tree_depth	integer	-		$[1, \infty)$
disable_progbar	logical	-	TRUE, FALSE	-
jit_compile	logical	-	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBotorchFullyBayesian`

Active bindings

`marshaled` (`logical(1)`) Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrBotorchFullyBayesian$new()`
- `LearnerRegrBotorchFullyBayesian$marshal()`
- `LearnerRegrBotorchFullyBayesian$unmarshal()`
- `LearnerRegrBotorchFullyBayesian$clone()`

`LearnerRegrBotorchFullyBayesian$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBotorchFullyBayesian$new()
```

`LearnerRegrBotorchFullyBayesian$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrBotorchFullyBayesian$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `marshal_model()`.

`LearnerRegrBotorchFullyBayesian$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrBotorchFullyBayesian$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `unmarshal_model()`.

`LearnerRegrBotorchFullyBayesian$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBotorchFullyBayesian$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Marc Becker

`mlr_learners_regr.botorch_mixedsingletaskgp`
BoTorch MixedSingleTaskGP Regression Learner

Description

Gaussian Process via botorch and gpytorch, using the `MixedSingleTaskGP`. See here for more details. Uses **reticulate** to interface with Python.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.botorch_mixedsingletaskgp")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **reticulate**

Parameters

Id	Type	Default	Levels
device	character	cpu	cpu, cuda
input_transform	character	-	normalize, standardize, log10, warp, none
outcome_transform	character	-	standardize, log, none

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBotorchMixedSingleTaskGP`

Active bindings

`marshaled` (`logical(1)`) Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrBotorchMixedSingleTaskGP$new()`
- `LearnerRegrBotorchMixedSingleTaskGP$marshal()`
- `LearnerRegrBotorchMixedSingleTaskGP$unmarshal()`
- `LearnerRegrBotorchMixedSingleTaskGP$clone()`

`LearnerRegrBotorchMixedSingleTaskGP$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBotorchMixedSingleTaskGP$new()
```

`LearnerRegrBotorchMixedSingleTaskGP$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrBotorchMixedSingleTaskGP$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrBotorchMixedSingleTaskGP$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrBotorchMixedSingleTaskGP$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrBotorchMixedSingleTaskGP$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBotorchMixedSingleTaskGP$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Marc Becker

`mlr_learners_regr.botorch_singletaskgp`
BoTorch SingleTaskGP Regression Learner

Description

Gaussian Process via botorch and gpytorch, using the `SingleTaskGP` model. See here for more details. Uses **reticulate** to interface with Python.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.botorch_singletaskgp")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **reticulate**

Parameters

Id	Type	Default	Levels
device	character	cpu	cpu, cuda
kernel	character	-	matern_2.5, matern_1.5, matern_0.5, rbf, linear, polynomial, periodic, cosine, rq,
input_transform	character	-	normalize, standardize, log10, warp, none
outcome_transform	character	-	standardize, log, none

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBotorchSingleTaskGP`

Active bindings

`marshaled` (`logical(1)`) Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrBotorchSingleTaskGP$new()`
- `LearnerRegrBotorchSingleTaskGP$marshal()`
- `LearnerRegrBotorchSingleTaskGP$unmarshal()`
- `LearnerRegrBotorchSingleTaskGP$clone()`

`LearnerRegrBotorchSingleTaskGP$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBotorchSingleTaskGP$new()
```

`LearnerRegrBotorchSingleTaskGP$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrBotorchSingleTaskGP$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `marshal_model()`.

`LearnerRegrBotorchSingleTaskGP$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrBotorchSingleTaskGP$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `unmarshal_model()`.

`LearnerRegrBotorchSingleTaskGP$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBotorchSingleTaskGP$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

Marc Becker

`mlr_learners_regr.brnn`

Regression Neural Network Learner

Description

Bayesian regularization for feed-forward neural networks. Calls `brnn::brnn()` from **brnn**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.brnn")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **brnn**

Parameters

Id	Type	Default	Levels	Range
change	numeric	0.001		$(-\infty, \infty)$
cores	integer	1		$[1, \infty)$
epochs	integer	1000		$[1, \infty)$
min_grad	numeric	1e-10		$(-\infty, \infty)$
Monte_Carlo	logical	FALSE	TRUE, FALSE	-
mu	numeric	0.005		$(-\infty, \infty)$
mu_dec	numeric	0.1		$(-\infty, \infty)$
mu_inc	numeric	10		$(-\infty, \infty)$
mu_max	numeric	1e+10		$(-\infty, \infty)$
neurons	integer	2		$[1, \infty)$
normalize	logical	TRUE	TRUE, FALSE	-
samples	integer	40		$[1, \infty)$
tol	numeric	1e-06		$(-\infty, \infty)$
verbose	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBrnn`

Methods

Public methods:

- `LearnerRegrBrnn$new()`
- `LearnerRegrBrnn$clone()`

`LearnerRegrBrnn$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBrnn$new()
```

`LearnerRegrBrnn$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBrnn$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

annanzrv

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.brnn")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.bst` *Regression Gradient Boosting Learner*

Description

Regression gradient boosting learner. Calls `bst::bst()` from **bst**.

Initial parameter values

- `Learner` = `"ls"`: Default base learner type
- `xval` = `0`: No cross-validation
- `maxdepth` = `1`: Maximum tree depth

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.bst")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **bst**, **rpart**

Parameters

Id	Type	Default	Levels	Range
center	logical	FALSE	TRUE, FALSE	-
coefir	untyped	NULL		-
cost	numeric	0.5		$[0, 1]$
cp	numeric	0.01		$[0, 1]$
df	integer	4		$[1, \infty)$
family	character	gaussian	gaussian, laplace, huber, rhuberDC, thingeDC, tbinomDC, binomdDC	-

f.init	untyped	NULL		-
fk	untyped	NULL		-
intercept	logical	TRUE	TRUE, FALSE	-
iter	integer	1		$[1, \infty)$
Learner	character	ls	ls, sm, tree	-
maxdepth	integer	1		$[1, 30]$
maxsurrogate	integer	5		$[0, \infty)$
minbucket	integer	-		$[1, \infty)$
minsplit	integer	20		$[1, \infty)$
mstop	integer	50		$[1, \infty)$
numsample	integer	50		$[1, \infty)$
nu	numeric	0.1		$[0, 1]$
q	numeric	-		$[0, 1]$
qh	numeric	-		$[0, 1]$
s	numeric	-		$[0, \infty)$
sh	numeric	-		$[0, \infty)$
start	logical	FALSE	TRUE, FALSE	-
surrogatestyle	integer	0		$[0, 1]$
threshold	character	adaptive	adaptive, fixed	-
trace	logical	FALSE	TRUE, FALSE	-
trun	logical	FALSE	TRUE, FALSE	-
twinboost	logical	FALSE	TRUE, FALSE	-
twintype	integer	1		$[1, 2]$
xselect.init	untyped	NULL		-
xval	integer	10		$[0, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBst`

Methods

Public methods:

- `LearnerRegrBst$new()`
- `LearnerRegrBst$clone()`

`LearnerRegrBst$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBst$new()
```

`LearnerRegrBst$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBst$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.bst")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.btgp`

Regression Bayesian Treed Gaussian Process Learner

Description

Bayesian treed Gaussian process regression model. Calls `tgp::btgp()` from **tgp**. For the predicted mean `ZZ.km` and for the predicted variance `ZZ.ks2` are chosen.

Factor features are one-hot encoded with reference encoding before fitting. If factors are present, `basemax` is set to the number of non-factor features so that tree proposals account for the numeric part of the design.

Initial parameter values

- `verb` is initialized to `0` to silence printing.
- `pred.n` is initialized to `FALSE` to skip prediction during training.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.btgp")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **tgp**

Parameters

Id	Type	Default	Levels	Range
meanfn	character	linear	constant, linear	-
bprior	character	bflat	b0, b0not, bflat, bmle, bmznot, bmzt	-
corr	character	expsep	exp, expsep, matern, sim	-
tree	untyped	c(0.5, 2)		-
BTE	untyped	c(2000L, 7000L, 2L)		-
R	integer	1		$[1, \infty)$
m0r1	logical	TRUE	TRUE, FALSE	-
linburn	logical	FALSE	TRUE, FALSE	-
itemps	untyped	NULL		-
pred.n	logical	-	TRUE, FALSE	-
krige	logical	TRUE	TRUE, FALSE	-
zcov	logical	FALSE	TRUE, FALSE	-
Ds2x	logical	FALSE	TRUE, FALSE	-
improv	logical	FALSE	TRUE, FALSE	-
nu	numeric	1.5		$(-\infty, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
verb	integer	-		$[0, 4]$
MAP	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBtgp`

Methods

Public methods:

- `LearnerRegrBtgp$new()`
- `LearnerRegrBtgp$clone()`

`LearnerRegrBtgp$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBtgp$new()
```

`LearnerRegrBtgp$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBtgp$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Gramacy RB (2007). "tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models." *Journal of Statistical Software*, **19**(9), 1–46. doi:10.18637/jss.v019.i09.

Gramacy RB, Taddy M (2010). "Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with tgp Version 2, an R Package for Treed Gaussian Process Models." *Journal of Statistical Software*, **33**(6), 1–48. doi:10.18637/jss.v033.i06.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.

- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.btgp")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.btgpllm`

Regression Bayesian Treed Gaussian Process LLM Learner

Description

Bayesian treed Gaussian process with jumps to the limiting linear model. Calls `tgp::btgpllm()` from **tgp**. For the predicted mean ZZ.km and for the predicted variance ZZ.ks2 are chosen.

Factor features are one-hot encoded with reference encoding before fitting. If factors are present, `basemax` is set to the number of non-factor features so that tree proposals account for the numeric part of the design.

Initial parameter values

- `verb` is initialized to `0` to silence printing.
- `pred.n` is initialized to `FALSE` to skip prediction during training.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.btgpllm")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **tgp**

Parameters

Id	Type	Default	Levels	Range
bprior	character	bflat	b0, b0not, bflat, bmle, bmznot, bmzt	-
corr	character	expsep	exp, expsep, matern, sim	-
Ds2x	logical	FALSE	TRUE, FALSE	-
improv	logical	FALSE	TRUE, FALSE	-
itemps	untyped	NULL		-
krige	logical	TRUE	TRUE, FALSE	-
linburn	logical	FALSE	TRUE, FALSE	-
m0r1	logical	TRUE	TRUE, FALSE	-
MAP	logical	TRUE	TRUE, FALSE	-
meanfn	character	linear	constant, linear	-
nu	numeric	1.5		$(-\infty, \infty)$
pred.n	logical	-	TRUE, FALSE	-
R	integer	1		$[1, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
verb	integer	-		$[0, 4]$
zcov	logical	FALSE	TRUE, FALSE	-
tree	untyped	c(0.5, 2)		-
gamma	untyped	c(10, 0.2, 0.7)		-
BTE	untyped	c(2000L, 7000L, 2L)		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBtgpllm`

Methods

Public methods:

- `LearnerRegrBtgpllm$new()`

- `LearnerRegrBtgpllm$clone()`

`LearnerRegrBtgpllm$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBtgpllm$new()
```

`LearnerRegrBtgpllm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBtgpllm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Gramacy RB (2007). "tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models." *Journal of Statistical Software*, **19**(9), 1–46. doi:10.18637/jss.v019.i09.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.btgpllm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)
```

```
# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.btlm`

Regression Bayesian Treed Linear Model Learner

Description

Bayesian treed linear model regression. Calls `tgp::btlm()` from **tgp**. For the predicted mean `ZZ.km` and for the predicted variance `ZZ.ks2` are chosen.

Factor features are one-hot encoded with reference encoding before fitting. If factors are present, `basemax` is set to the number of non-factor features so that tree proposals account for the numeric part of the design.

Initial parameter values

- `verb` is initialized to `0` to silence printing.
- `pred.n` is initialized to `FALSE` to skip prediction during training.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.btlm")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **tgp**

Parameters

Id	Type	Default	Levels	Range
meanfn	character	linear	constant, linear	-
bprior	character	bflat	b0, b0not, bflat, bmle, bmznot, bmzt	-
tree	untyped	c(0.5, 2)		-
BTE	untyped	c(2000L, 7000L, 2L)		-
R	integer	1		$[1, \infty)$
m0r1	logical	TRUE	TRUE, FALSE	-
itemps	untyped	NULL		-
pred.n	logical	-	TRUE, FALSE	-
krige	logical	TRUE	TRUE, FALSE	-
zcov	logical	FALSE	TRUE, FALSE	-
Ds2x	logical	FALSE	TRUE, FALSE	-
improv	logical	FALSE	TRUE, FALSE	-
trace	logical	FALSE	TRUE, FALSE	-
verb	integer	-		$[0, 4]$
MAP	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrBtlm`

Methods

Public methods:

- `LearnerRegrBtlm$new()`
- `LearnerRegrBtlm$clone()`

`LearnerRegrBtlm$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrBtlm$new()
```

`LearnerRegrBtlm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrBtlm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Gramacy RB (2007). "tgp: An R Package for Bayesian Nonstationary, Semiparametric Nonlinear Regression and Design by Treed Gaussian Process Models." *Journal of Statistical Software*, **19**(9), 1–46. doi:10.18637/jss.v019.i09.

Gramacy RB, Taddy M (2010). "Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with tgp Version 2, an R Package for Treed Gaussian Process Models." *Journal of Statistical Software*, **33**(6), 1–48. doi:10.18637/jss.v033.i06.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.btlm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.catboost`

Gradient Boosted Decision Trees Regression Learner

Description

Gradient boosting algorithm that also supports categorical data. Calls `catboost::catboost.train()` from package 'catboost'.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.catboost")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **catboost**

Parameters

Id	Type	Default	Levels
loss_function	character	RMSE	MAE, MAPE, Poisson, Quantile, RMSE, LogLinQuantile,
learning_rate	numeric	0.03	
random_seed	integer	0	
l2_leaf_reg	numeric	3	
bootstrap_type	character	-	Bayesian, Bernoulli, MVS, Poisson, No
bagging_temperature	numeric	1	
subsample	numeric	-	
sampling_frequency	character	PerTreeLevel	PerTree, PerTreeLevel
sampling_unit	character	Object	Object, Group
mvs_reg	numeric	-	
random_strength	numeric	1	
depth	integer	6	
grow_policy	character	SymmetricTree	SymmetricTree, Depthwise, Lossguide
min_data_in_leaf	integer	1	
max_leaves	integer	31	
has_time	logical	FALSE	TRUE, FALSE
rsm	numeric	1	
nan_mode	character	Min	Min, Max
fold_permutation_block	integer	-	
leaf_estimation_method	character	-	Newton, Gradient, Exact

leaf_estimation_iterations	integer	-	
leaf_estimation_backtracking	character	AnyImprovement	No, AnyImprovement, Armijo
fold_len_multiplier	numeric	2	
approx_on_full_history	logical	TRUE	TRUE, FALSE
boosting_type	character	-	Ordered, Plain
boost_from_average	logical	-	TRUE, FALSE
langevin	logical	FALSE	TRUE, FALSE
diffusion_temperature	numeric	10000	
score_function	character	Cosine	Cosine, L2, NewtonCosine, NewtonL2
monotone_constraints	untyped	-	
feature_weights	untyped	-	
first_feature_use_penalties	untyped	-	
penalties_coefficient	numeric	1	
per_object_feature_penalties	untyped	-	
model_shrink_rate	numeric	-	
model_shrink_mode	character	-	Constant, Decreasing
target_border	numeric	-	
border_count	integer	-	
feature_border_type	character	GreedyLogSum	Median, Uniform, UniformAndQuantiles, MaxLogSum, Mi
per_float_feature_quantization	untyped	-	
thread_count	integer	1	
task_type	character	CPU	CPU, GPU
devices	untyped	-	
logging_level	character	Silent	Silent, Verbose, Info, Debug
metric_period	integer	1	
train_dir	untyped	"catboost_info"	
model_size_reg	numeric	0.5	
allow_writing_files	logical	FALSE	TRUE, FALSE
save_snapshot	logical	FALSE	TRUE, FALSE
snapshot_file	untyped	-	
snapshot_interval	integer	600	
simple_ctr	untyped	-	
combinations_ctr	untyped	-	
ctr_target_border_count	integer	-	
counter_calc_method	character	Full	SkipTest, Full
max_ctr_complexity	integer	-	
ctr_leaf_count_limit	integer	-	
store_all_simple_ctr	logical	FALSE	TRUE, FALSE
final_ctr_computation_mode	character	Default	Default, Skip
verbose	logical	FALSE	TRUE, FALSE
ntree_start	integer	0	
ntree_end	integer	0	
early_stopping_rounds	integer	-	
eval_metric	untyped	-	
use_best_model	logical	-	TRUE, FALSE
iterations	integer	1000	

Installation

See https://catboost.ai/en/docs/concepts/r-installation.

Initial parameter values

- logging_level:
  - Actual default: "Verbose"
  - Adjusted default: "Silent"
  - Reason for change: consistent with other mlr3 learners
- thread_count:
  - Actual default: -1
  - Adjusted default: 1
  - Reason for change: consistent with other mlr3 learners
- allow_writing_files:
  - Actual default: TRUE
  - Adjusted default: FALSE
  - Reason for change: consistent with other mlr3 learners
- save_snapshot:
  - Actual default: TRUE
  - Adjusted default: FALSE
  - Reason for change: consistent with other mlr3 learners

Early stopping

Early stopping can be used to find the optimal number of boosting rounds. Set early_stopping_rounds to an integer value to monitor the performance of the model on the validation set while training. For information on how to configure the validation set, see the *Validation* section of mlr3::Learner.

Super classes

mlr3::Learner -> mlr3::LearnerRegr -> LearnerRegrCatboost

Active bindings

internal_valid_scores The validation scores for the eval_metric and the loss_function, evaluated on the internal validation data with all tree_count trees, i.e. the trees that are also used for prediction.

best_valid_scores The validation scores of the best iteration. Because use_best_model truncates the model to the best iteration, these are identical to $internal_valid_scores whenever early stopping is activated. If early stopping is not activated or use_best_model is FALSE, no best iteration is tracked and this is an empty list.

internal_tuned_values Returns the early stopped iterations if early_stopping_rounds was set during training.

`validate` How to construct the internal validation data. This parameter can be either `NULL`, a ratio, `"test"`, or `"predefined"`.

Methods

Public methods:

- `LearnerRegrCatboost$new()`
- `LearnerRegrCatboost$importance()`
- `LearnerRegrCatboost$clone()`

`LearnerRegrCatboost$new()`: Create a `LearnerRegrCatboost` object.

Usage:

```
LearnerRegrCatboost$new()
```

`LearnerRegrCatboost$importance()`: The importance scores are calculated using `catboost::catboost.get_feature` setting `type = "FeatureImportance"`, returned for 'all'.

Usage:

```
LearnerRegrCatboost$importance()
```

Returns: Named `numeric()`.

`LearnerRegrCatboost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrCatboost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sumny

References

Dorogush, Veronika A, Ershov, Vasily, Gulin, Andrey (2018). "CatBoost: gradient boosting with categorical features support." *arXiv preprint arXiv:1810.11363*.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.catboost")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.cforest

Regression Conditional Random Forest Learner

Description

A random forest based on conditional inference trees (`partykit::ctree()`). Calls `partykit::cforest()` from **partykit**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.cforest")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **partykit**, **sandwich**, **coin**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
replace	logical	FALSE	TRUE, FALSE	-
fraction	numeric	0.632		$[0, 1]$
mtry	integer	-		$[0, \infty)$
mtryratio	numeric	-		$[0, 1]$
applyfun	untyped	-		-
cores	integer	NULL		$(-\infty, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
cluster	untyped	-		-
scores	untyped	-		-
teststat	character	quadratic	quadratic, maximum	-
splitstat	character	quadratic	quadratic, maximum	-
splittest	logical	FALSE	TRUE, FALSE	-
testtype	character	Univariate	Bonferroni, MonteCarlo, Univariate, Teststatistic	-
nmax	untyped	-		-
pargs	untyped	-		-
alpha	numeric	0.05		$[0, 1]$
mincriterion	numeric	0		$[0, 1]$
logmincriterion	numeric	0		$(-\infty, \infty)$
minsplit	integer	20		$[1, \infty)$
minbucket	integer	7		$[1, \infty)$
minprob	numeric	0.01		$[0, 1]$
stump	logical	FALSE	TRUE, FALSE	-
lookahead	logical	FALSE	TRUE, FALSE	-
MIA	logical	FALSE	TRUE, FALSE	-
maxvar	integer	-		$[1, \infty)$
nresample	integer	9999		$[1, \infty)$
tol	numeric	1.490116e-08		$[0, \infty)$
maxsurrogate	integer	0		$[0, \infty)$
numsurrogate	logical	FALSE	TRUE, FALSE	-
maxdepth	integer	Inf		$[0, \infty)$
multiway	logical	FALSE	TRUE, FALSE	-
splittry	integer	2		$[0, \infty)$
intersplit	logical	FALSE	TRUE, FALSE	-
majority	logical	FALSE	TRUE, FALSE	-
caseweights	logical	TRUE	TRUE, FALSE	-
saveinfo	logical	FALSE	TRUE, FALSE	-
update	logical	FALSE	TRUE, FALSE	-
splitflavour	character	ctree	ctree, exhaustive	-
OOB	logical	FALSE	TRUE, FALSE	-
simplify	logical	TRUE	TRUE, FALSE	-
scale	logical	TRUE	TRUE, FALSE	-
nperm	integer	1		$[0, \infty)$
risk	character	loglik	loglik, misclassification	-
conditional	logical	FALSE	TRUE, FALSE	-

threshold	numeric	0.2	$(-\infty, \infty)$

Custom mlr3 parameters

- `mtry`:
  - This hyperparameter can alternatively be set via the added hyperparameter `mtryratio` as `mtry = max(ceiling(mtryratio * n_features), 1)`. Note that `mtry` and `mtryratio` are mutually exclusive.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrCForest`

Methods

Public methods:

- `LearnerRegrCForest$new()`
- `LearnerRegrCForest$oob_error()`
- `LearnerRegrCForest$clone()`

`LearnerRegrCForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrCForest$new()
```

`LearnerRegrCForest$oob_error()`: The out-of-bag error, calculated using the OOB predictions from `partykit`.

Usage:

```
LearnerRegrCForest$oob_error()
```

Returns: `numeric(1)`.

`LearnerRegrCForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrCForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sumny

References

Hothorn T, Zeileis A (2015). "partykit: A Modular Toolkit for Recursive Partytioning in R." *Journal of Machine Learning Research*, **16**(118), 3905-3909. http://jmlr.org/papers/v16/hothorn15a.html.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. doi:10.1198/106186006x133933. https://doi.org/10.1198/106186006x133933.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.cforest")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.crs` *Regression Splines Learner*

Description

Categorical Regression Splines. Calls `crs::crs()` from **crs**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.crs")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **crs**

Parameters

Id	Type	Default	Levels	Range
degree	integer	3		$[0, \infty)$
segments	integer	1		$[1, \infty)$
include	integer	-		$(-\infty, \infty)$
lambda	untyped	-		-
lambda.discrete	logical	FALSE	TRUE, FALSE	-
lambda.discrete.num	integer	100		$[0, \infty)$
cv	character	nomad	nomad, exhaustive, none	-
cv.threshold	integer	1000		$[0, \infty)$
cv.func	character	cv.ls	cv.ls, cv.gcv, cv.aic	-
kernel	logical	TRUE	TRUE, FALSE	-
degree.max	integer	10		$[0, \infty)$
segments.max	integer	10		$[1, \infty)$
degree.min	integer	0		$[0, \infty)$
segments.min	integer	1		$[1, \infty)$
cv.df.min	integer	1		$(-\infty, \infty)$
complexity	character	degree-knots	degree-knots, degree, knots	-
knots	character	quantiles	quantiles, uniform, auto	-
basis	character	auto	auto, additive, tensor, glp	-
prune	logical	FALSE	TRUE, FALSE	-
restarts	integer	0		$[0, \infty)$
nmulti	integer	5		$[0, \infty)$
singular.ok	logical	FALSE	TRUE, FALSE	-

deriv	integer	0		$[0, \infty)$
data.return	logical	FALSE	TRUE, FALSE	-
model.return	logical	FALSE	TRUE, FALSE	-
random.seed	integer	-		$(-\infty, \infty)$
tau	numeric	-		$[0, 1]$
initial.mesh.size.real	untyped	-		-
initial.mesh.size.integer	untyped	-		-
max.bb.eval	untyped	-		-
max.eval	untyped	-		-
min.mesh.size.real	untyped	-		-
min.mesh.size.integer	untyped	-		-
min.frame.size.real	untyped	-		-
min.frame.size.integer	untyped	-		-
display.nomad.progress	logical	TRUE	TRUE, FALSE	-
display.warnings	logical	TRUE	TRUE, FALSE	-
opts	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrCrs`

Methods

Public methods:

- `LearnerRegrCrs$new()`
- `LearnerRegrCrs$clone()`

`LearnerRegrCrs$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrCrs$new()
```

`LearnerRegrCrs$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrCrs$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.crs", cv = "none")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.ctree`

Regression Conditional Inference Tree Learner

Description

Regression Partition Tree where a significance test is used to determine the univariate splits. Calls `partykit::ctree()` from **partykit**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.ctree")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **partykit**, **sandwich**, **coin**

Parameters

Id	Type	Default	Levels	Range
teststat	character	quadratic	quadratic, maximum	-
splitstat	character	quadratic	quadratic, maximum	-
splittest	logical	FALSE	TRUE, FALSE	-
testtype	character	Bonferroni	Bonferroni, MonteCarlo, Univariate, Teststatistic	-
nmax	untyped	-		-
alpha	numeric	0.05		$[0, 1]$
mincriterion	numeric	0.95		$[0, 1]$
logmincriterion	numeric	-		$(-\infty, \infty)$
minsplit	integer	20		$[1, \infty)$
minbucket	integer	7		$[1, \infty)$
minprob	numeric	0.01		$[0, \infty)$
stump	logical	FALSE	TRUE, FALSE	-
lookahead	logical	FALSE	TRUE, FALSE	-
MIA	logical	FALSE	TRUE, FALSE	-
maxvar	integer	-		$[1, \infty)$
nresample	integer	9999		$[1, \infty)$
tol	numeric	-		$[0, \infty)$
maxsurrogate	integer	0		$[0, \infty)$
numsurrogate	logical	FALSE	TRUE, FALSE	-
mtry	integer	Inf		$[0, \infty)$
maxdepth	integer	Inf		$[0, \infty)$
multiway	logical	FALSE	TRUE, FALSE	-
splittry	integer	2		$[0, \infty)$
intersplit	logical	FALSE	TRUE, FALSE	-
majority	logical	FALSE	TRUE, FALSE	-
caseweights	logical	FALSE	TRUE, FALSE	-
applyfun	untyped	-		-
cores	integer	NULL		$(-\infty, \infty)$
saveinfo	logical	TRUE	TRUE, FALSE	-
update	logical	FALSE	TRUE, FALSE	-
splitflavour	character	ctree	ctree, exhaustive	-
cluster	untyped	-		-

scores	untyped	-		-
doFit	logical	TRUE	TRUE, FALSE	-
maxpts	integer	25000		$(-\infty, \infty)$
abseps	numeric	0.001		$[0, \infty)$
releps	numeric	0		$[0, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrCTree`

Methods

Public methods:

- `LearnerRegrCTree$new()`
- `LearnerRegrCTree$clone()`

`LearnerRegrCTree$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrCTree$new()
```

`LearnerRegrCTree$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrCTree$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sumny

References

Hothorn T, Zeileis A (2015). "partykit: A Modular Toolkit for Recursive Partytioning in R." *Journal of Machine Learning Research*, **16**(118), 3905-3909. http://jmlr.org/papers/v16/hothorn15a.html.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. doi:10.1198/106186006x133933. https://doi.org/10.1198/106186006x133933.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.ctree")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.cubist`

Regression Cubist Learner

Description

Rule-based model that is an extension of Quinlan's M5 model tree. Each tree contains linear regression models at the terminal leaves. Calls `Cubist::cubist()` from **Cubist**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.cubist")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3extralearners**, **Cubist**

Parameters

Id	Type	Default	Levels	Range
committees	integer	-		$[1, 100]$
unbiased	logical	FALSE	TRUE, FALSE	-
rules	integer	100		$[1, \infty)$
extrapolation	numeric	100		$[0, 100]$
sample	integer	0		$[0, \infty)$
seed	integer	-		$(-\infty, \infty)$
label	untyped	"outcome"		-
neighbors	integer	-		$[0, 9]$
strip_time_stamps	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrCubist`

Methods

Public methods:

- `LearnerRegrCubist$new()`
- `LearnerRegrCubist$clone()`

`LearnerRegrCubist$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrCubist$new()
```

`LearnerRegrCubist$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrCubist$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sumny

References

Quinlan, R J, others (1992). "Learning with continuous classes." In *5th Australian joint conference on artificial intelligence*, volume 92, 343–348. World Scientific.

Quinlan, Ross J (1993). "Combining instance-based and model-based learning." In *Proceedings of the tenth international conference on machine learning*, 236–243.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.cubist")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.cv_ncvreg`

Cross-Validated MCP- or SCAD-penalized Regression Learner

Description

MCP- or SCAD-penalized regression models with k-fold cross validation for choosing the regularization parameter lambda. Calls `ncvreg::cv.ncvreg()` from package **ncvreg** with `family` set to `"gaussian"` (default). Poisson family is also supported. Covariates are internally standardized by `ncvreg` prior to model fitting.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.cv_ncvreg")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **ncvreg**

Parameters

Id	Type	Default	Levels	Range
family	character	gaussian	gaussian, poisson	-
penalty	character	MCP	MCP, SCAD, lasso	-
gamma	numeric	-		$[0, \infty)$
alpha	numeric	1		$[0, 1]$
lambda.min	numeric	-		$[0, \infty)$
nlambda	integer	100		$[1, \infty)$
lambda	untyped	-		-
eps	numeric	1e-04		$(-\infty, \infty)$
max.iter	integer	10000		$[1, \infty)$
convex	logical	TRUE	TRUE, FALSE	-
dfmax	integer	-		$[0, \infty)$
penalty.factor	untyped	-		-
warn	logical	TRUE	TRUE, FALSE	-
returnX	logical	TRUE	TRUE, FALSE	-
cluster	untyped	-		-
nfolds	integer	10		$(-\infty, \infty)$
fold	untyped	-		-
returnY	logical	FALSE	TRUE, FALSE	-

trace	logical	FALSE	TRUE, FALSE	-
pred_lambda	numeric	-		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrNCVreg`

Methods

Public methods:

- `LearnerRegrNCVreg$new()`
- `LearnerRegrNCVreg$selected_features()`
- `LearnerRegrNCVreg$clone()`

`LearnerRegrNCVreg$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrNCVreg$new()
```

`LearnerRegrNCVreg$selected_features()`: Returns the set of selected features which have non-zero coefficients. Calls the internal `coef.cv.ncvreg()` function.

Usage:

```
LearnerRegrNCVreg$selected_features(lambda = NULL)
```

Arguments:

`lambda` (`numeric(1)`)
Custom `lambda`, defaults to the lambda with the minimum CV error.

Returns: (`character()`) vector of feature names.

`LearnerRegrNCVreg$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrNCVreg$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bblodfon

References

Breheny P, Huang J (2011). "Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection." *Annals of Applied Statistics*, **5**(1), 232–253. ISSN 1932-6157. doi:10.1214/10AOAS388.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.cv_ncvreg")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.decision_stump`

Regression Decision Stump Learner

Description

Decision Stump Learner. Calls `RWeka::DecisionStump()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.decision_stump")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrDecisionStump`

Active bindings

`marshaled (logical(1))`
: Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrDecisionStump$new()`
- `LearnerRegrDecisionStump$marshal()`
- `LearnerRegrDecisionStump$unmarshal()`
- `LearnerRegrDecisionStump$clone()`

`LearnerRegrDecisionStump$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrDecisionStump$new()
```

`LearnerRegrDecisionStump$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrDecisionStump$marshal(...)
```

Arguments:

`...` (any)
: Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrDecisionStump$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrDecisionStump$unmarshal(...)
```

Arguments:

`...` (any)
: Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrDecisionStump$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrDecisionStump$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.decision_stump")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.decision_table`

Regression Decision Table Learner

Description

Simple Decision Table majority regressor. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Initial parameter values

- `E`:
  - Has only 2 out of 4 original evaluation measures : rmse and mae with rmse being the default
  - Reason for change: this learner should only contain evaluation measures appropriate for regression tasks

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- `P_best`:
  - original id: P
- `D_best`:
  - original id: D
- `N_best`:
  - original id: N
- `S_best`:
  - original id: S
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.decision_table")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
S	character	BestFirst	BestFirst, GreedyStepwise	-
X	integer	1		$(-\infty, \infty)$
E	character	rmse	rmse, mae	-
I	logical	-	TRUE, FALSE	-
R	logical	-	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
P_best	untyped	-		-
D_best	character	1	0, 1, 2	-
N_best	integer	-		$(-\infty, \infty)$
S_best	integer	1		$(-\infty, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrDecisionTable`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrDecisionTable$new()`
- `LearnerRegrDecisionTable$marshal()`
- `LearnerRegrDecisionTable$unmarshal()`
- `LearnerRegrDecisionTable$clone()`

`LearnerRegrDecisionTable$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrDecisionTable$new()
```

`LearnerRegrDecisionTable$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrDecisionTable$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrDecisionTable$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrDecisionTable$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrDecisionTable$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrDecisionTable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Kohavi R (1995). "The Power of Decision Tables." In *8th European Conference on Machine Learning*, 174–189.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.decision_table")
print(learner)

# Define a Task
```

```
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.earth`

Implementation of Multivariate Adaptive Regression Splines

Description

This is an alternative implementation of MARS (Multivariate Adaptive Regression Splines). MARS is trademarked and thus not used as the name. The name "earth" stands for "Enhanced Adaptive Regression Through Hinges".

Calls `earth::earth()` from **earth**.

Details

Methods for variance estimations are not yet implemented.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.earth")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “se”
- Feature Types: “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3extralearners**, **earth**

Parameters

Id	Type	Default	Levels	Range
wp	untyped	NULL		-
offset	untyped	NULL		-
keepxy	logical	FALSE	TRUE, FALSE	-
trace	character	0	0, .3, .5, 1, 2, 3, 4, 5	-
degree	integer	1		$[1, \infty)$
penalty	numeric	2		$[-1, \infty)$
nk	untyped	NULL		-
thresh	numeric	0.001		$(-\infty, \infty)$
minspan	numeric	0		$[0, \infty)$
endspan	numeric	0		$[0, \infty)$
newvar.penalty	numeric	0		$[0, \infty)$
fast.k	integer	20		$[0, \infty)$
fast.beta	integer	1		$[0, 1]$
linpreds	untyped	FALSE		-
allowed	untyped	-		-
pmethod	character	backward	backward, none, exhaustive, forward, seqrep, cv	-
nprune	integer	-		$[0, \infty)$
nfold	integer	0		$[0, \infty)$
ncross	integer	1		$[0, \infty)$
stratify	logical	TRUE	TRUE, FALSE	-
varmod.method	character	none	none, const, lm, rlm, earth, gam, power, power0, x.lm, x.rlm, [...]	-
varmod.exponent	numeric	1		$(-\infty, \infty)$
varmod.conv	numeric	1		$[0, 1]$
varmod.clamp	numeric	0.1		$(-\infty, \infty)$
varmod.minspan	numeric	-3		$(-\infty, \infty)$
Scale.y	logical	FALSE	TRUE, FALSE	-
Adjust.endspan	numeric	2		$(-\infty, \infty)$
Auto.linpreds	logical	TRUE	TRUE, FALSE	-
Force.weights	logical	FALSE	TRUE, FALSE	-
Use.beta.cache	logical	TRUE	TRUE, FALSE	-
Force.xtx.prune	logical	FALSE	TRUE, FALSE	-
Get.leverages	logical	TRUE	TRUE, FALSE	-
Exhaustive.tol	numeric	1e-10		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrEarth`

Methods

Public methods:

- `LearnerRegrEarth$new()`

- `LearnerRegrEarth$clone()`

`LearnerRegrEarth$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrEarth$new()
```

`LearnerRegrEarth$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrEarth$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

pkopper

References

Milborrow, Stephen, Hastie, T, Tibshirani, R (2014). "Earth: multivariate adaptive regression spline models." *R package version*, **3**(7).

Friedman, H J (1991). "Multivariate adaptive regression splines." *The annals of statistics*, **19**(1), 1–67.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.earth")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)
```

```
# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.evtree`
Regression Decision Tree Learner

Description

Evolutionary learning of globally optimal regression trees. Calls `evtree::evtree()` from**evtree**.

Initial parameter values

`pmutatemajor`, `pmutateminor`, `pcrossover`, `psplit`, and `pprune`, are scaled internally to sum to 100.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrEvtree`

Methods

Public methods:

- `LearnerRegrEvtree$new()`
- `LearnerRegrEvtree$clone()`

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

Usage:

```
LearnerRegrEvtree$new()
```

`LearnerRegrEvtree$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrEvtree$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

`mlr_learners_regr.exhaustive_search`

Linear Regression with Exhaustive Search

Description

Linear Regression, where the features used in the model are selected by running Exhaustive Search.

Calls `ExhaustiveSearch::ExhaustiveSearch()` from **ExhaustiveSearch**.

Initial parameter values

- `family`:
  - Actual default: NULL
  - Adjusted default: "gaussian"
  - Reason for change: To comply with mlr3 architecture, we differentiate between classification and regression learners.
- `nThreads`:
  - Actual default: NULL
  - Adjusted default: 1
  - Reason for change: Suppressing the automatic internal parallelization if `cv.folds` > 0.
- `quietly`:
  - Actual default: FALSE
  - Adjusted default: TRUE
  - Reason for change: Suppression of constant printing to console

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.exhaustive_search")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **ExhaustiveSearch**

Parameters

Id	Type	Default	Levels	Range
family	character	-	gaussian, binomial	-
performanceMeasure	character	-	MSE, AIC	-
combsUpTo	integer	-		$[1, \infty)$
nResults	integer	5000		$[1, \infty)$
nThreads	integer	-		$[1, \infty)$
testSetIDs	integer	-		$[1, \infty)$
errorVal	untyped	-1		-
quietly	logical	-	TRUE, FALSE	-
checkLarge	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrExhaustiveSearch`

Methods

Public methods:

- `LearnerRegrExhaustiveSearch$new()`
- `LearnerRegrExhaustiveSearch$selected_features()`
- `LearnerRegrExhaustiveSearch$clone()`

`LearnerRegrExhaustiveSearch$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrExhaustiveSearch$new()
```

`LearnerRegrExhaustiveSearch$selected_features()`: Extracts selected features of this learner.

Usage:

```
LearnerRegrExhaustiveSearch$selected_features()
```

`LearnerRegrExhaustiveSearch$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrExhaustiveSearch$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ngerman

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# define learner
learner = lrn("regr.exhaustive_search", predict_type = "se")

# define task
tsk_cars = tsk("mtcars")

# train learner
learner$train(tsk_cars)

# extract selected features
learner$selected_features()

# predict on training task
learner$predict(tsk_cars)
```

`mlr_learners_regr.fnn` *Regression Fast Nearest Neighbor Search Learner*

Description

Fast Nearest Neighbor Regression. Calls `FNN::knn.reg()` from **FNN**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.fnn")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **FNN**

Parameters

Id	Type	Default	Levels	Range
k	integer	1		$[1, \infty)$
algorithm	character	kd_tree	kd_tree, cover_tree, brute	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrFNN`

Methods

Public methods:

- `LearnerRegrFNN$new()`
- `LearnerRegrFNN$clone()`

`LearnerRegrFNN$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrFNN$new()
```

`LearnerRegrFNN$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrFNN$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Boltz, Sylvain, Debreuve, Eric, Barlaud, Michel (2007). "kNN-based high-dimensional Kullback-Leibler distance for tracking." In *Eighth International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS'07)*, 16–16. IEEE.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.fnn")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.frbs`

Regression Fuzzy Rule-based System Learner

Description

Fuzzy rule-based regression learner supporting multiple inference engines such as WM, HYFIS, ANFIS, DENFIS, SBC, and several genetic fuzzy systems. Calls `frbs::frbs.learn()` from **frbs**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.frbs")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **frbs**

Parameters

Id	Type	Default	Levels
alpha.heuristic	numeric	1	
d	numeric	2	
Dthr	numeric	0.1	
eps.high	numeric	0.5	
eps.low	numeric	0.15	
epsilon	numeric	0.9	
max.gen	integer	10	
max.iter	integer	10	
max.tune	integer	10	
method.type	character	WM	WM, SBC, HYFIS, ANFIS, DENFIS, FS.HGD, FIR.DM, GFS.FR.MOGU
mode.tuning	character	GLOBAL	GLOBAL, LOCAL
num.labels	integer	7	
persen_cross	numeric	1	
persen_mutant	numeric	1	
popu.size	integer	10	
r.a	numeric	0.5	
range.data	untyped	NULL	
rule.selection	logical	FALSE	TRUE, FALSE
step.size	numeric	0.01	
type.defuz	character	WAM	WAM, FIRST.MAX, LAST.MAX, MEAN.MAX, COG

type.implication.func	character	ZADEH	DIENES_RESHER, LUKASIEWICZ, ZADEH, GOGUEN, GODEL, SH
type.mf	character	GAUSSIAN	TRIANGLE, TRAPEZOID, GAUSSIAN, SIGMOID, BELL
type.snorm	character	MAX	MAX, HAMACHER, YAGER, PRODUCT, BOUNDED
type.tnorm	character	MIN	MIN, HAMACHER, YAGER, PRODUCT, BOUNDED

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrFrbs`

Methods

Public methods:

- `LearnerRegrFrbs$new()`
- `LearnerRegrFrbs$clone()`

`LearnerRegrFrbs$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrFrbs$new()
```

`LearnerRegrFrbs$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrFrbs$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Riza LS, Bergmeir C, Herrera F, Benitez JM (2015). "frbs: Fuzzy Rule-Based Systems for Classification and Regression in R." *Journal of Statistical Software*, **65**(6), 1–30. https://www.jstatsoft.org/v65/i06/.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.

- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.frbs")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.gam *Regression Generalized Additive Model Learner*

Description

Generalized additive models. Calls `mgcv::gam()` from package **mgcv**.

Formula

A gam formula specific to the task at hand is required for the `formula` parameter (see example and `?mgcv::formula.gam`). Beware, if no formula is provided, a fallback formula is used that will make the gam behave like a glm (this behavior is required for the unit tests). Only features specified in the formula will be used, superseding columns with col_roles "feature" in the task.

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated during training via the `offset` argument in `mgcv::gam()`. No offset is applied during prediction for this learner.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.gam")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “se”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3extralearners**, **mgcv**

Parameters

Id	Type	Default	Levels	Range
family	character	gaussian	gaussian, poisson	-
formula	untyped	-		-
method	character	GCV.Cp	GCV.Cp, GACV.Cp, REML, P-REML, ML, P-ML	-
optimizer	untyped	c("outer", "newton")		-
scale	numeric	0		$(-\infty, \infty)$
select	logical	FALSE	TRUE, FALSE	-
knots	untyped	NULL		-
sp	untyped	NULL		-
min.sp	untyped	NULL		-
H	untyped	NULL		-
gamma	numeric	1		$[1, \infty)$
paraPen	untyped	NULL		-
G	untyped	NULL		-
in.out	untyped	NULL		-
drop.unused.levels	logical	TRUE	TRUE, FALSE	-
drop.intercept	logical	FALSE	TRUE, FALSE	-
nei	untyped	-		-
nthreads	integer	1		$[1, \infty)$
irls.reg	numeric	0		$[0, \infty)$
epsilon	numeric	1e-07		$[0, \infty)$
maxit	integer	200		$(-\infty, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
mgcv.tol	numeric	1e-07		$[0, \infty)$
mgcv.half	integer	15		$[0, \infty)$
rank.tol	numeric	1.490116e-08		$[0, \infty)$
nlm	untyped	list()		-
optim	untyped	list()		-
newton	untyped	list()		-
idLinksBases	logical	TRUE	TRUE, FALSE	-
scalePenalty	logical	TRUE	TRUE, FALSE	-
efs.lspmax	integer	15		$[0, \infty)$
efs.tol	numeric	0.1		$[0, \infty)$

scale.est	character	fletcher	fletcher, pearson, deviance	-
ncv.threads	integer	1		$[1, \infty)$
edge.correct	logical	FALSE	TRUE, FALSE	-
block.size	integer	1000		$(-\infty, \infty)$
unconditional	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrGam`

Methods

Public methods:

- `LearnerRegrGam$new()`
- `LearnerRegrGam$clone()`

`LearnerRegrGam$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrGam$new()
```

`LearnerRegrGam$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrGam$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

pierrecamilleri

References

Hastie, J T, Tibshirani, J R (2017). *Generalized additive models*. Routledge.

Wood, Simon (2012). "mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML smoothness estimation."

Examples

```
# simple example
t = tsk("mtcars")
l = lrn("regr.gam")
l$param_set$values$formula = mpg ~ cyl + am + s(disp) + s(hp)
l$train(t)
l$model
```

`mlr_learners_regr.gamboost`
Boosted Generalized Additive Regression Learner

Description

Fit a generalized additive regression model using a boosting algorithm. Calls `mboost::gamboost()` from **mboost**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.gamboost")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **mboost**

Parameters

Id	Type	Default	Levels	
baselearner	character	bbs	bbs, bols, btree	-
dfbase	integer	4		(
family	character	Gaussian	Gaussian, Laplace, Huber, Poisson, GammaReg, NBinomial, Hurdle, custom	-
custom.family	untyped	-		-
nuirange	untyped	c(0, 100)		-
d	numeric	NULL		(
mstop	integer	100		(
nu	numeric	0.1		(
risk	character	inbag	inbag, oobag, none	-
oobweights	untyped	NULL		-
trace	logical	FALSE	TRUE, FALSE	-
stopintern	untyped	FALSE		-
na.action	untyped	stats::na.omit		-

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated via the `offset` argument in `mboost`'s training function. No offset is applied during prediction for this learner.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrGAMBoost`

Methods

Public methods:

- `LearnerRegrGAMBoost$new()`
- `LearnerRegrGAMBoost$clone()`

`LearnerRegrGAMBoost$new()`: Create a `LearnerRegrGAMBoost` object.

Usage:

```
LearnerRegrGAMBoost$new()
```

`LearnerRegrGAMBoost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrGAMBoost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Bühlmann, Peter, Yu, Bin (2003). "Boosting with the L 2 loss: regression and classification." *Journal of the American Statistical Association*, **98**(462), 324–339.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
lrn("regr.gamboost")
```

`mlr_learners_regr.gaussian_processes`
Regression Gaussian Processes Learner From Weka

Description

Gaussian Processes. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- `E_poly`:
  - original id: E
- `L_poly`:
  - original id: L (duplicated L for when K is set to PolyKernel)
- `C_poly`:
  - original id: C
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern
- `output-debug-info` for kernel parameter removed:
  - enables debugging output (if available) to be printed
- Reason for change: The parameter is removed because it's unclear how to actually use it.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.gaussian_processes")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels
subset	untyped	-	
na.action	untyped	-	
L	numeric	1	
N	character	0	0, 1, 2
K	character	supportVector.PolyKernel	supportVector.NormalizedPolyKernel, supportVector.Pc
S	integer	1	
E_poly	numeric	1	
L_poly	logical	FALSE	TRUE, FALSE
C_poly	integer	250007	
output_debug_info	logical	FALSE	TRUE, FALSE
do_not_check_capabilities	logical	FALSE	TRUE, FALSE
num_decimal_places	integer	2	
batch_size	integer	100	
options	untyped	NULL	

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrGaussianProcesses`

Active bindings

`marshaled (logical(1))`
: Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrGaussianProcesses$new()`
- `LearnerRegrGaussianProcesses$marshal()`
- `LearnerRegrGaussianProcesses$unmarshal()`
- `LearnerRegrGaussianProcesses$clone()`

`LearnerRegrGaussianProcesses$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrGaussianProcesses$new()
```

`LearnerRegrGaussianProcesses$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrGaussianProcesses$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrGaussianProcesses$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrGaussianProcesses$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrGaussianProcesses$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrGaussianProcesses$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Mackay DJ (1998). "Introduction to Gaussian Processes."

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.gaussian_processes")
print(learner)

# Define a Task
task = tsk("mtcars")
```

```
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.gausspr`

Regression Gaussian Process Learner

Description

Gaussian process for regression. Calls `kernlab::gausspr()` from **kernlab**. Parameters `sigma`, `degree`, `scale`, `offset` and `order` are added to make tuning `kpar` easier. If `kpar` is provided then these new parameters are ignored. If none are provided then the default "automatic" is used for `kpar`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.gausspr")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3extralearners**, **kernlab**

Parameters

Id	Type	Default	Levels
scaled	untyped	TRUE	
kernel	character	rbfdot	rbfdot, polydot, vanilladot, tanhdot, laplacedot, besseldot, anovadot, splinedot
sigma	numeric	-	

degree	numeric	-	
scale	numeric	-	
offset	numeric	-	
order	numeric	-	
kpar	untyped	"automatic"	
var	numeric	0.001	
variance.model	logical	FALSE	TRUE, FALSE
tol	numeric	0.001	
fit	logical	TRUE	TRUE, FALSE
na.action	untyped	na.omit	

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrGausspr`

Methods

Public methods:

- `LearnerRegrGausspr$new()`
- `LearnerRegrGausspr$clone()`

`LearnerRegrGausspr$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrGausspr$new()
```

`LearnerRegrGausspr$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrGausspr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Karatzoglou, Alexandros, Smola, Alex, Hornik, Kurt, Zeileis, Achim (2004). "kernlab-an S4 package for kernel methods in R." *Journal of statistical software*, **11**(9), 1–20.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.gausspr")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.gbm *Regression Gradient Boosting Machine Learner*

Description

Gradient Boosting Regression Algorithm. Calls `gbm::gbm()` from **gbm**.

Weights are ignored for quantile prediction.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.gbm")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "quantiles"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **gbm**

Parameters

Id	Type	Default	Levels	Range
distribution	character	gaussian	gaussian, laplace, poisson, tdist	-
n.trees	integer	100		$[1, \infty)$
interaction.depth	integer	1		$[1, \infty)$
n.minobsinnode	integer	10		$[1, \infty)$
shrinkage	numeric	0.001		$[0, \infty)$
bag.fraction	numeric	0.5		$[0, 1]$
train.fraction	numeric	1		$[0, 1]$
cv.folds	integer	0		$(-\infty, \infty)$
keep.data	logical	FALSE	TRUE, FALSE	-
verbose	logical	FALSE	TRUE, FALSE	-
n.cores	integer	1		$(-\infty, \infty)$
var.monotone	untyped	-		-

Parameter changes

- `keep.data`:
  - Actual default: TRUE
  - Adjusted default: FALSE
  - Reason for change: `keep.data` = `FALSE` saves memory during model fitting.
- `n.cores`:
  - Actual default: NULL
  - Adjusted default: 1
  - Reason for change: Suppressing the automatic internal parallelization if `cv.folds` > 0.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrGBM`

Methods

Public methods:

- `LearnerRegrGBM$new()`
- `LearnerRegrGBM$importance()`
- `LearnerRegrGBM$clone()`

`LearnerRegrGBM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrGBM$new()
```

`LearnerRegrGBM$importance()`: The importance scores are extracted by `gbm::relative.influence()` from the model.

Usage:

```
LearnerRegrGBM$importance()
```

Returns: Named `numeric()`.

`LearnerRegrGBM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrGBM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Friedman, H J (2002). "Stochastic gradient boosting." *Computational statistics & data analysis*, **38**(4), 367–378.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
lrn("regr.gbm")
```

`mlr_learners_regr.glm` *Generalized Linear Regression*

Description

Generalized linear model. Calls `stats::glm()` from base package 'stats'. For logistic regression please use `mlr_learners_classif.log_reg`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.glm")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, 'stats'

Parameters

Id	Type	Default	Levels	Range
singular.ok	logical	TRUE	TRUE, FALSE	-
x	logical	FALSE	TRUE, FALSE	-
y	logical	TRUE	TRUE, FALSE	-
model	logical	TRUE	TRUE, FALSE	-
etastart	untyped	-		-
mustart	untyped	-		-
start	untyped	NULL		-
family	character	gaussian	gaussian, poisson, quasipoisson, Gamma, inverse.gaussian	-
na.action	character	-	na.omit, na.pass, na.fail, na.exclude	-
link	character	-	logit, probit, cauchit, cloglog, identity, log, sqrt, 1/mu^2, inverse	-
epsilon	numeric	1e-08		$(-\infty, \infty)$
maxit	numeric	25		$(-\infty, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
dispersion	untyped	NULL		-
type	character	link	response, link, terms	-
use_pred_offset	logical	TRUE	TRUE, FALSE	-

Initial parameter values

- `type`
  - Actual default: `"link"`
  - Adjusted default: `"response"`
  - Reason for change: Response scale more natural for predictions.

Offset

If a `Task` has a column with the role `offset`, it will automatically be used during training. The offset is incorporated through the formula interface to ensure compatibility with `stats::glm()`. We add it to the model formula as `offset(<column_name>)` and also include it in the training data. During prediction, the default behavior is to use the offset column from the test set (enabled by `use_pred_offset = TRUE`). Otherwise, if the user sets `use_pred_offset = FALSE`, a zero offset is applied, effectively disabling the offset adjustment during prediction.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrGlm`

Methods

Public methods:

- `LearnerRegrGlm$new()`
- `LearnerRegrGlm$clone()`

`LearnerRegrGlm$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrGlm$new()
```

`LearnerRegrGlm$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrGlm$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

salauer

References

Hosmer Jr, W D, Lemeshow, Stanley, Sturdivant, X R (2013). *Applied logistic regression*, volume 398. John Wiley & Sons.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.glm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.glmboost

Boosted Generalized Linear Regression Learner

Description

Fit a generalized linear regression model using a boosting algorithm. Calls `mboost::glmboost()` from **mboost**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.glmboost")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **mboost**

Parameters

Id	Type	Default	Levels	
family	character	Gaussian	Gaussian, Laplace, Huber, Poisson, GammaReg, NBinomial, Hurdle, custom	-
custom.family	untyped	-		-
nuirange	untyped	c(0, 100)		-
d	numeric	NULL		(
center	logical	TRUE	TRUE, FALSE	-
mstop	integer	100		(
nu	numeric	0.1		(
risk	character	inbag	inbag, oobag, none	-
oobweights	untyped	NULL		-
trace	logical	FALSE	TRUE, FALSE	-
stopintern	untyped	FALSE		-
na.action	untyped	stats::na.omit		-
contrasts.arg	untyped	-		-

Offset

If a Task contains a column with the offset role, it is automatically incorporated via the offset argument in mboost's training function. No offset is applied during prediction for this learner.

Super classes

mlr3::Learner -> mlr3::LearnerRegr -> LearnerRegrGLMBoost

Methods

Public methods:

- LearnerRegrGLMBoost$new()
- LearnerRegrGLMBoost$clone()

`LearnerRegrGLMBoost$new()`: Create a `LearnerRegrGLMBoost` object.

Usage:

```
LearnerRegrGLMBoost$new()
```

`LearnerRegrGLMBoost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrGLMBoost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Bühlmann, Peter, Yu, Bin (2003). "Boosting with the L 2 loss: regression and classification." *Journal of the American Statistical Association*, **98**(462), 324–339.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.glmboost")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.gpfit`

Regression Gaussian Process Learner (GPfit)

Description

Gaussian process regression via `GPfit::GP_fit()` from **GPfit**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.gpfit")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **GPfit**

Parameters

Id	Type	Default	Levels	Range
control	untyped	NULL		-
nug_thres	numeric	-		$[0, \infty)$
trace	logical	-	TRUE, FALSE	-
maxit	integer	-		$[1, \infty)$
optim_start	untyped	-		-
scale	logical	-	TRUE, FALSE	-
type	character	exponential	exponential, matern	-
matern_nu_k	integer	0		$[0, \infty)$
power	numeric	1.95		$[1, 2]$

Super classes

mlr3::Learner -> mlr3::LearnerRegr -> LearnerRegrGPfit

Methods

Public methods:

- LearnerRegrGPfit$new()
- LearnerRegrGPfit$clone()

LearnerRegrGPfit$new(): Creates a new instance of this R6 class.

Usage:

```
LearnerRegrGPfit$new()
```

LearnerRegrGPfit$clone(): The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrGPfit$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Note

As the optimization routine assumes that the inputs are scaled to the unit hypercube, the input gets scaled for each variable by default. If this is not wanted, scale = FALSE`` has to be set. We replace the GPfit paramet = list(type = 'exponential',power = 1.95)to be separate parameterstypeandpower, in the case of corr = list(type = 'matern', nu = 0.5), the separate parameters are typeandmatern_nu_k = 0, and nu is computed by nu = (2 * matern_nu_k + 1) / 2 = 0.5‘.

Author(s)

awinterstetter

References

MacDonald, B., Ranjan, P., Chipman, H. (2015). "GPfit: An R package for fitting a Gaussian process model to deterministic simulator outputs." *Journal of Statistical Software*, **64**, 1–23. ISSN 1548-7660. doi:10.18637/jss.v064.i12.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.

- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.gpfit", maxit = 1)
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.grf *Generalized Random Forest Regression Learner*

Description

Generalized random forest for regression. Calls `grf::regression_forest()` from **grf**.

Custom mlr3 parameters

- `sample.weights` is not exposed as a hyperparameter. Observation weights are taken from the task instead, see the `weights` property.
- `estimate.variance` is not exposed as a hyperparameter. It is enabled automatically when the `predict_type` is `"se"`, which requires `ci.group.size` to be at least 2.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.grf")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **grf**

Parameters

Id	Type	Default	Levels	Range
num.trees	integer	2000		$[1, \infty)$
clusters	untyped	NULL		-
equalize.cluster.weights	logical	FALSE	TRUE, FALSE	-
sample.fraction	numeric	0.5		$[0, 1]$
mtry	integer	-		$[1, \infty)$
min.node.size	integer	5		$[1, \infty)$
honesty	logical	TRUE	TRUE, FALSE	-
honesty.fraction	numeric	0.5		$[0, 1]$
honesty.prune.leaves	logical	TRUE	TRUE, FALSE	-
alpha	numeric	0.05		$[0, 0.25]$
imbalance.penalty	numeric	0		$[0, \infty)$
ci.group.size	integer	2		$[1, \infty)$
tune.parameters	untyped	"none"		-
tune.num.trees	integer	50		$[1, \infty)$
tune.num.reps	integer	100		$[1, \infty)$
tune.num.draws	integer	1000		$[1, \infty)$
compute.oob.predictions	logical	TRUE	TRUE, FALSE	-
seed	integer	-		$(-\infty, \infty)$
linear.correction.variables	untyped	NULL		-
ll.lambda	numeric	NULL		$[0, \infty)$
ll.weight.penalty	logical	FALSE	TRUE, FALSE	-
num.threads	integer	-		$[1, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrGRF`

Methods

Public methods:

- `LearnerRegrGRF$new()`
- `LearnerRegrGRF$clone()`

`LearnerRegrGRF$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrGRF$new()
```

`LearnerRegrGRF$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrGRF$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

jesuisantoine

References

Athey, Susan, Tibshirani, Julie, Wager, Stefan (2019). "Generalized random forests." *The Annals of Statistics*, **47**(2), 1148–1178. doi:10.1214/18AOS1709.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.grf")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.h2o.deeplearning`
Regression H2O Deep Learning Learner

Description

Regression feed-forward multilayer artificial neural network learner. Class `h2o::h2o.deeplearning()` from package **h2o**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.h2o.deeplearning")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **h2o**

Parameters

Id	Type	Default	Levels
activation	character	Rectifier	Rectifier, Tanh, TanhWithDropout, RectifierWithDropout,
adaptive_rate	logical	TRUE	TRUE, FALSE
autoencoder	logical	FALSE	TRUE, FALSE
average_activation	numeric	0	
categorical_encoding	character	AUTO	AUTO, Enum, OneHotInternal, OneHotExplicit, Binary, E
checkpoint	untyped	NULL	
diagnostics	logical	TRUE	TRUE, FALSE
distribution	character	AUTO	AUTO, gaussian, poisson, gamma, tweedie, laplace, huber
elastic_averaging	logical	FALSE	TRUE, FALSE
elastic_averaging_moving_rate	numeric	0.9	
elastic_averaging_regularization	numeric	0.001	
epochs	numeric	10	
epsilon	numeric	1e-08	

export_checkpoints_dir	untyped	NULL	
export_weights_and_biases	logical	FALSE	TRUE, FALSE
fast_mode	logical	TRUE	TRUE, FALSE
force_load_balance	logical	TRUE	TRUE, FALSE
hidden	untyped	c(200L, 200L)	
hidden_dropout_ratios	numeric	0.5	
huber_alpha	numeric	0.9	
ignore_const_cols	logical	TRUE	TRUE, FALSE
initial_weight_distribution	character	UniformAdaptive	UniformAdaptive, Uniform, Normal
initial_weight_scale	numeric	1	
input_dropout_ratio	numeric	0	
l1	numeric	0	
l2	numeric	0	
loss	character	Automatic	Automatic, Quantile, Quadratic, Absolute, Huber
max_categorical_features	integer	2147483647	
max_runtime_secs	numeric	0	
max_w2	numeric	3.402823e+38	
mini_batch_size	integer	1	
missing_values_handling	character	MeanImputation	MeanImputation, Skip
momentum_ramp	numeric	1e+06	
momentum_stable	numeric	0	
momentum_start	numeric	0	
nesterov_accelerated_gradient	logical	TRUE	TRUE, FALSE
overwrite_with_best_model	logical	TRUE	TRUE, FALSE
pretrained_autoencoder	untyped	NULL	
quantile_alpha	numeric	0.5	
quiet_mode	logical	FALSE	TRUE, FALSE
rate	numeric	0.005	
rate_annealing	numeric	1e-06	
rate_decay	numeric	1	
regression_stop	numeric	1e-06	
replicate_training_data	logical	TRUE	TRUE, FALSE
reproducible	logical	FALSE	TRUE, FALSE
rho	numeric	0.99	
score_duty_cycle	numeric	0.1	
score_each_iteration	logical	FALSE	TRUE, FALSE
score_interval	numeric	5	
score_training_samples	integer	10000	
score_validation_samples	integer	0	
seed	integer	-1	
shuffle_training_data	logical	FALSE	TRUE, FALSE
single_node_mode	logical	FALSE	TRUE, FALSE
sparse	logical	FALSE	TRUE, FALSE
sparsity_beta	numeric	0	
standardize	logical	TRUE	TRUE, FALSE
stopping_metric	character	AUTO	AUTO, deviance, MSE, RMSE, MAE, RMSLE
stopping_rounds	integer	5	
stopping_tolerance	numeric	0	

target_ratio_comm_to_comp	numeric	0.05	
train_samples_per_iteration	integer	-2	
tweedie_power	numeric	1.5	
verbose	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrH2ODeeplearning`

Methods

Public methods:

- `LearnerRegrH2ODeeplearning$new()`
- `LearnerRegrH2ODeeplearning$clone()`

`LearnerRegrH2ODeeplearning$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrH2ODeeplearning$new()
```

`LearnerRegrH2ODeeplearning$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrH2ODeeplearning$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Fryda T, LeDell E, Gill N, Aiello S, Fu A, Candel A, Click C, Kraljevic T, Nykodym T, Aboyoun P, Kurka M, Malohlava M, Poirier S, Wong W (2025). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.46.0.9, https://github.com/h2oai/h2o-3.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners

- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.h2o.deeplearning")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.h2o.gbm`

Regression H2O GBM Learner

Description

Gradient boosting machine learner. Class `h2o::h2o.gbm()` from package **h2o**.

H2O Connection

If no running H2O connection is found, the learner will automatically start a local H2O server on `127.0.0.1` via `h2o::h2o.init()`. If you want to connect to a remote H2O cluster, call `h2o::h2o.init()` with the appropriate arguments before training or predicting.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.h2o.gbm")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **h2o**

Parameters

Id	Type	Default	Levels
auto_rebalance	logical	TRUE	TRUE, FALSE
build_tree_one_node	logical	FALSE	TRUE, FALSE
categorical_encoding	character	AUTO	AUTO, Enum, OneHotInternal, OneHotExplicit, Binary, Eige
check_constant_response	logical	TRUE	TRUE, FALSE
checkpoint	untyped	NULL	
col_sample_rate	numeric	1	
col_sample_rate_change_per_level	numeric	1	
col_sample_rate_per_tree	numeric	1	
custom_distribution_func	untyped	NULL	
distribution	character	gaussian	AUTO, poisson, laplace, tweedie, gaussian, huber, gamma, qu
export_checkpoints_dir	untyped	NULL	
histogram_type	character	AUTO	AUTO, UniformAdaptive, Random, QuantilesGlobal, RoundI
huber_alpha	numeric	0.9	
ignore_const_cols	logical	TRUE	TRUE, FALSE
in_training_checkpoints_dir	untyped	NULL	
in_training_checkpoints_tree_interval	integer	1	
interaction_constraints	untyped	NULL	
learn_rate	numeric	0.1	
learn_rate_annealing	numeric	1	
max_abs_leafnode_pred	numeric	Inf	
max_depth	integer	5	
max_runtime_secs	numeric	0	
min_rows	integer	10	
min_split_improvement	numeric	1e-05	
monotone_constraints	untyped	NULL	
nbins	integer	20	
nbins_cats	integer	1024	
nbins_top_level	integer	1024	
ntrees	integer	50	
pred_noise_bandwidth	numeric	0	
quantile_alpha	numeric	0.5	
sample_rate	numeric	1	
score_each_iteration	logical	FALSE	TRUE, FALSE
score_tree_interval	integer	0	
seed	integer	-1	
stopping_metric	character	AUTO	AUTO, deviance, MSE, RMSE, MAE, RMSLE
stopping_rounds	integer	0	
stopping_tolerance	numeric	0.001	

tweedie_power	numeric	1.5	
verbose	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrH2OGBM`

Methods

Public methods:

- `LearnerRegrH2OGBM$new()`
- `LearnerRegrH2OGBM$clone()`

`LearnerRegrH2OGBM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrH2OGBM$new()
```

`LearnerRegrH2OGBM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrH2OGBM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Fryda T, LeDell E, Gill N, Aiello S, Fu A, Candel A, Click C, Kraljevic T, Nykodym T, Aboyoun P, Kurka M, Malohlava M, Poirier S, Wong W (2025). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.46.0.9, https://github.com/h2oai/h2o-3.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.h2o.gbm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.h2o.glm

Generalized Linear Regression H2O Learner

Description

Generalized linear model. Calls `h2o::h2o.glm()` from package **h2o**.

H2O Connection

If no running H2O connection is found, the learner will automatically start a local H2O server on `127.0.0.1` via `h2o::h2o.init()`. If you want to connect to a remote H2O cluster, call `h2o::h2o.init()` with the appropriate arguments before training or predicting.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.h2o.glm")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **h2o**

Parameters

Id	Type	Default	Levels
alpha	numeric	0.5	
beta_constraints	untyped	NULL	
beta_epsilon	numeric	1e-04	
build_null_model	logical	FALSE	TRUE, FALSE
calc_like	logical	FALSE	TRUE, FALSE
checkpoint	untyped	NULL	
cold_start	logical	FALSE	TRUE, FALSE
compute_p_values	logical	FALSE	TRUE, FALSE
early_stopping	logical	TRUE	TRUE, FALSE
export_checkpoints_dir	untyped	NULL	
generate_scoring_history	logical	FALSE	TRUE, FALSE
generate_variable_inflation_factors	logical	FALSE	TRUE, FALSE
gradient_epsilon	numeric	-1	
HGLM	logical	FALSE	TRUE, FALSE
ignore_const_cols	logical	TRUE	TRUE, FALSE
interactions	untyped	NULL	
interaction_pairs	untyped	NULL	
intercept	logical	TRUE	TRUE, FALSE
lambda	numeric	1e-05	
lambda_min_ratio	numeric	-1	
lambda_search	logical	FALSE	TRUE, FALSE
link	character	family_default	family_default, identity, log, inverse
max_active_predictors	integer	-1	
max_iterations	integer	-1	
max_runtime_secs	numeric	0	
missing_values_handling	character	MeanImputation	MeanImputation, Skip, PlugValues
nlambdas	integer	-1	
non_negative	logical	FALSE	TRUE, FALSE
objective_epsilon	numeric	-1	
obj_reg	numeric	-1	
plug_values	untyped	NULL	
random_columns	untyped	NULL	
rand_family	untyped	NULL	
rand_link	untyped	NULL	
remove_collinear_columns	logical	FALSE	TRUE, FALSE
score_each_iteration	logical	FALSE	TRUE, FALSE
score_iteration_interval	integer	-1	
seed	integer	-1	
solver	character	AUTO	AUTO, IRLSM, L_BFGS, COORDINATE_DESCENT_
standardize	logical	TRUE	TRUE, FALSE
startval	untyped	NULL	
stopping_metric	character	AUTO	AUTO, deviance, MSE, RMSE, MAE, RMSLE
stopping_rounds	integer	0	
stopping_tolerance	numeric	0.001	

Model family

This learner calls h2o::h2o.glm() with family = "gaussian"

Super classes

mlr3::Learner -> mlr3::LearnerRegr -> LearnerRegrH2OGLM

Methods

Public methods:

- LearnerRegrH2OGLM$new()
- LearnerRegrH2OGLM$clone()

LearnerRegrH2OGLM$new(): Creates a new instance of this R6 class.

Usage:

```
LearnerRegrH2OGLM$new()
```

LearnerRegrH2OGLM$clone(): The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrH2OGLM$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

awinterstetter

References

Fryda T, LeDell E, Gill N, Aiello S, Fu A, Candel A, Click C, Kraljevic T, Nykodym T, Aboyoun P, Kurka M, Malohlava M, Poirier S, Wong W (2025). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.46.0.9, https://github.com/h2oai/h2o-3.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.h2o.glm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.h2o.randomForest

Regression H2O Random Forest Learner

Description

Regression random forest learner. Calls `h2o::h2o.randomForest()` from package **h2o**.

H2O Connection

If no running H2O connection is found, the learner will automatically start a local H2O server on `127.0.0.1` via `h2o::h2o.init()`. If you want to connect to a remote H2O cluster, call `h2o::h2o.init()` with the appropriate arguments before training or predicting.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.h2o.randomForest")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **h2o**

Parameters

Id	Type	Default	Levels
build_tree_one_node	logical	FALSE	TRUE, FALSE
categorical_encoding	character	AUTO	AUTO, Enum, OneHotInternal, OneHotExplicit, Binary, Eigen, L
check_constant_response	logical	TRUE	TRUE, FALSE
checkpoint	untyped	NULL	
col_sample_rate_change_per_level	numeric	1	
col_sample_rate_per_tree	numeric	1	
export_checkpoints_dir	untyped	NULL	
histogram_type	character	AUTO	AUTO, UniformAdaptive, Random, QuantilesGlobal, RoundRobi
ignore_const_cols	logical	TRUE	TRUE, FALSE
max_depth	integer	20	
max_runtime_secs	numeric	0	
min_rows	numeric	1	
min_split_improvement	numeric	1e-05	
mtries	integer	-1	
nbins	integer	20	
nbins_cats	integer	1024	
nbins_top_level	integer	1024	
ntrees	integer	50	
sample_rate	numeric	0.632	
score_each_iteration	logical	FALSE	TRUE, FALSE
score_tree_interval	integer	0	
seed	integer	-1	
stopping_metric	character	AUTO	AUTO, deviance, MSE, RMSE, MAE, RMSLE
stopping_rounds	integer	0	
stopping_tolerance	numeric	0.001	
verbose	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrH2ORandomForest`

Methods

Public methods:

- `LearnerRegrH2ORandomForest$new()`
- `LearnerRegrH2ORandomForest$clone()`

`LearnerRegrH2ORandomForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrH2ORandomForest$new()
```

`LearnerRegrH2ORandomForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrH2ORandomForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Fryda T, LeDell E, Gill N, Aiello S, Fu A, Candel A, Click C, Kraljevic T, Nykodym T, Aboyoun P, Kurka M, Malohlava M, Poirier S, Wong W (2025). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.46.0.9, https://github.com/h2oai/h2o-3.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.h2o.randomForest")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)
```

```
# Score the predictions
predictions$score()
```

`mlr_learners_regr.IBk` *Regression IBk Learner*

Description

Instance based algorithm: K-nearest neighbors regression. Calls `RWeka::IBk()` from **RWeka**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.IBk")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **RWeka**

Parameters

Id	Type	Default	Levels
subset	untyped	-	
na.action	untyped	-	
weight	character	-	I, F
K	integer	1	
E	logical	FALSE	TRUE, FALSE
W	integer	0	
X	logical	FALSE	TRUE, FALSE
A	character	LinearNNSearch	BallTree, CoverTree, FilteredNeighbourSearch, KDTree, Linearl
output_debug_info	logical	FALSE	TRUE, FALSE
do_not_check_capabilities	logical	FALSE	TRUE, FALSE
num_decimal_places	integer	2	
batch_size	integer	100	
options	untyped	NULL	

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern
- `weight`:
  - original id: I and F
- Reason for change: original `I` and `F` params are interdependent (`I` can only be `TRUE` when `F` is `FALSE` and vice versa). The easiest way to encode this is to combine `I` and `F` into one factor param.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrIBk`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrIBk$new()`
- `LearnerRegrIBk$marshal()`
- `LearnerRegrIBk$unmarshal()`
- `LearnerRegrIBk$clone()`

`LearnerRegrIBk$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrIBk$new()
```

`LearnerRegrIBk$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrIBk$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrIBk$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrIBk$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrIBk$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrIBk$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

henrifnk

References

Aha, W D, Kibler, Dennis, Albert, K M (1991). "Instance-based learning algorithms." *Machine learning*, **6**(1), 37–66.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.IBk")
print(learner)

# Define a Task
task = tsk("mtcars")
```

```
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.kstar`
Regression KStar Learner

Description

Instance-based regressor which differs from other instance-based learners in that it uses an entropy-based distance function. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.kstar")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
B	integer	20		$(-\infty, \infty)$
E	logical	-	TRUE, FALSE	-
M	character	a	a, d, m, n	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrKStar`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrKStar$new()`
- `LearnerRegrKStar$marshal()`
- `LearnerRegrKStar$unmarshal()`
- `LearnerRegrKStar$clone()`

`LearnerRegrKStar$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrKStar$new()
```

`LearnerRegrKStar$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrKStar$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrKStar$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrKStar$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrKStar$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrKStar$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Cleary JG, Trigg LE (1995). "K*: An Instance-based Learner Using an Entropic Distance Measure." In *12th International Conference on Machine Learning*, 108-114.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.kstar")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.ksvm`

Regression Kernlab Support Vector Machine

Description

Support Vector Regression. Calls `kernlab::ksvm()` from **kernlab**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.ksvm")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **kernlab**

Parameters

Id	Type	Default	Levels	Range
scaled	logical	TRUE	TRUE, FALSE	-
type	character	eps-svr	eps-svr, nu-svr, eps-bsvr	-
kernel	character	rbfdot	rbfdot, polydot, vanilladot, laplacedot, besseldot, anovadot	-
C	numeric	1		$(-\infty, \infty)$
nu	numeric	0.2		$[0, \infty)$
epsilon	numeric	0.1		$(-\infty, \infty)$
cache	integer	40		$[1, \infty)$
tol	numeric	0.001		$[0, \infty)$
shrinking	logical	TRUE	TRUE, FALSE	-
sigma	numeric	-		$[0, \infty)$
degree	integer	-		$[1, \infty)$
scale	numeric	-		$[0, \infty)$
order	integer	-		$(-\infty, \infty)$
offset	numeric	-		$(-\infty, \infty)$
na.action	untyped	na.omit		-
fit	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrKSVM`

Methods

Public methods:

- `LearnerRegrKSVM$new()`
- `LearnerRegrKSVM$clone()`

`LearnerRegrKSVM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrKSVM$new()
```

`LearnerRegrKSVM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrKSVM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

mboecker

References

Karatzoglou, Alexandros, Smola, Alex, Hornik, Kurt, Zeileis, Achim (2004). "kernlab-an S4 package for kernel methods in R." *Journal of statistical software*, **11**(9), 1–20.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.ksvm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.laGP`
Regression Local Approximate Gaussian Process Learner

Description

Local approximate Gaussian process for regression. Calls `laGP::aGP()` from **laGP**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.laGP")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **laGP**

Parameters

Id	Type	Default	Levels	Range
start	integer	6		$[6, \infty)$
end	integer	50		$[6, \infty)$
d	untyped	NULL		-
g	untyped	1/10000		-
method	character	alc	alc, alcray, efi, mspe, nn	-
close	integer	-		$[0, \infty)$
numrays	integer	-		$[0, \infty)$
verb	numeric	0		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrLaGP`

Methods

Public methods:

- `LearnerRegrLaGP$new()`
- `LearnerRegrLaGP$clone()`

`LearnerRegrLaGP$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrLaGP$new()
```

`LearnerRegrLaGP$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrLaGP$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

Parameters `start` and `end` tune the initial and maximum neighborhood sizes used for the local GP fit, `d` can fix the length-scale parameters, `g` configures the nugget/regularization term, and `method` selects the search heuristic (`alc`, `alcray`, `efi`, `mspe`, `nn`). Arguments `close` and `numrays` further refine the ray-based search used by `method = "alcray"`, while `verb` controls the output verbosity.

Author(s)

awinterstetter

References

Gramacy, B. R (2016). "laGP: Large-Scale Spatial Modeling via Local Approximate Gaussian Processes in R." *Journal of Statistical Software*, **72**(1), 1–46. doi:10.18637/jss.v072.i01.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner (use a small neighborhood size for this task)
learner = lrn("regr.laGP", end = 15)
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.liblinear`
L2-Regularized Support Vector Regression Learner

Description

L2 regularized support vector regression. Calls `LiblineaR::LiblineaR()` from **LiblineaR**.

Details

Type of SVR depends on `type` argument:

- `type = 11` - L2-regularized L2-loss support vector regression (primal)
- `type = 12` – L2-regularized L2-loss support vector regression (dual)
- `type = 13` – L2-regularized L1-loss support vector regression (dual)

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.liblinear")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3extralearners**, **LiblineaR**

Parameters

Id	Type	Default	Levels	Range
type	integer	11		$[11, 13]$
cost	numeric	1		$[0, \infty)$
bias	numeric	1		$(-\infty, \infty)$
svr_eps	numeric	NULL		$[0, \infty)$
cross	integer	0		$[0, \infty)$
verbose	logical	FALSE	TRUE, FALSE	-
findC	logical	FALSE	TRUE, FALSE	-
useInitC	logical	TRUE	TRUE, FALSE	-

Initial parameter values

- `svr_eps`:
  - Actual default: `NULL`
  - Initial value: 0.001
  - Reason for change: `svr_eps` is type dependent and the "type" is handled by the mlr3learner. The default value is set to the default of the respective "type".

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrLiblineaR`

Methods

Public methods:

- `LearnerRegrLiblineaR$new()`
- `LearnerRegrLiblineaR$clone()`

`LearnerRegrLiblineaR$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrLiblineaR$new()
```

`LearnerRegrLiblineaR$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrLiblineaR$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Fan, Rong-En, Chang, Kai-Wei, Hsieh, Cho-Jui, Wang, Xiang-Rui, Lin, Chih-Jen (2008). "LIBLINEAR: A library for large linear classification." *the Journal of machine Learning research*, **9**, 1871–1874.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.liblinear")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.lightgbm`
Regression LightGBM Learner

Description

Gradient boosting algorithm. Calls `lightgbm::lightgbm()` from **lightgbm**. The list of parameters can be found here and in the documentation of `lightgbm::lgb.train()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.lightgbm")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **lightgbm**

Parameters

Id	Type	Default	Levels
objective	character	regression	regression, regression_l1, huber, fair, poisson, quantile, mape, gam
eval	untyped	-	
verbose	integer	1	
record	logical	TRUE	TRUE, FALSE
eval_freq	integer	1	
callbacks	untyped	-	
reset_data	logical	FALSE	TRUE, FALSE
boosting	character	gbdt	gbdt, rf, dart, goss
linear_tree	logical	FALSE	TRUE, FALSE
learning_rate	numeric	0.1	
num_leaves	integer	31	
tree_learner	character	serial	serial, feature, data, voting
num_threads	integer	0	
device_type	character	cpu	cpu, gpu
seed	integer	-	
deterministic	logical	FALSE	TRUE, FALSE
data_sample_strategy	character	bagging	bagging, goss
force_col_wise	logical	FALSE	TRUE, FALSE
force_row_wise	logical	FALSE	TRUE, FALSE
histogram_pool_size	integer	-1	

max_depth	integer	-1	
min_data_in_leaf	integer	20	
min_sum_hessian_in_leaf	numeric	0.001	
bagging_fraction	numeric	1	
bagging_freq	integer	0	
bagging_seed	integer	3	
bagging_by_query	logical	FALSE	TRUE, FALSE
feature_fraction	numeric	1	
feature_fraction_bynode	numeric	1	
feature_fraction_seed	integer	2	
extra_trees	logical	FALSE	TRUE, FALSE
extra_seed	integer	6	
max_delta_step	numeric	0	
lambda_l1	numeric	0	
lambda_l2	numeric	0	
linear_lambda	numeric	0	
min_gain_to_split	numeric	0	
drop_rate	numeric	0.1	
max_drop	integer	50	
skip_drop	numeric	0.5	
xgboost_dart_mode	logical	FALSE	TRUE, FALSE
uniform_drop	logical	FALSE	TRUE, FALSE
drop_seed	integer	4	
top_rate	numeric	0.2	
other_rate	numeric	0.1	
min_data_per_group	integer	100	
max_cat_threshold	integer	32	
cat_l2	numeric	10	
cat_smooth	numeric	10	
max_cat_to_onehot	integer	4	
top_k	integer	20	
monotone_constraints	untyped	NULL	
monotone_constraints_method	character	basic	basic, intermediate, advanced
monotone_penalty	numeric	0	
feature_contri	untyped	NULL	
forcedsplits_filename	untyped	""	
refit_decay_rate	numeric	0.9	
cegb_tradeoff	numeric	1	
cegb_penalty_split	numeric	0	
cegb_penalty_feature_lazy	untyped	-	
cegb_penalty_feature_coupled	untyped	-	
path_smooth	numeric	0	
interaction_constraints	untyped	-	
use_quantized_grad	logical	TRUE	TRUE, FALSE
num_grad_quant_bins	integer	4	
quant_train_renew_leaf	logical	FALSE	TRUE, FALSE
stochastic_rounding	logical	TRUE	TRUE, FALSE
serializable	logical	TRUE	TRUE, FALSE

max_bin	integer	255	
max_bin_by_feature	untyped	NULL	
min_data_in_bin	integer	3	
bin_construct_sample_cnt	integer	200000	
data_random_seed	integer	1	
is_enable_sparse	logical	TRUE	TRUE, FALSE
enable_bundle	logical	TRUE	TRUE, FALSE
use_missing	logical	TRUE	TRUE, FALSE
zero_as_missing	logical	FALSE	TRUE, FALSE
feature_pre_filter	logical	TRUE	TRUE, FALSE
pre_partition	logical	FALSE	TRUE, FALSE
two_round	logical	FALSE	TRUE, FALSE
forcedbins_filename	untyped	""	
boost_from_average	logical	TRUE	TRUE, FALSE
reg_sqrt	logical	FALSE	TRUE, FALSE
alpha	numeric	0.9	
fair_c	numeric	1	
poisson_max_delta_step	numeric	0.7	
tweedie_variance_power	numeric	1.5	
metric_freq	integer	1	
num_machines	integer	1	
local_listen_port	integer	12400	
time_out	integer	120	
machines	untyped	""	
gpu_platform_id	integer	-1	
gpu_device_id	integer	-1	
gpu_device_id_list	untyped	NULL	
gpu_use_dp	logical	FALSE	TRUE, FALSE
num_gpu	integer	1	
start_iteration_predict	integer	0	
num_iteration_predict	integer	-1	
pred_early_stop	logical	FALSE	TRUE, FALSE
pred_early_stop_freq	integer	10	
pred_early_stop_margin	numeric	10	
num_iterations	integer	100	
early_stopping_rounds	integer	-	
early_stopping_min_delta	numeric	-	
first_metric_only	logical	FALSE	TRUE, FALSE

Initial parameter values

- `num_threads`:
  - Actual default: 0L
  - Initial value: 1L
  - Reason for change: Prevents accidental conflicts with `future`.

- verbose:
  - Actual default: 1L
  - Initial value: -1L
  - Reason for change: Prevents accidental conflicts with mlr messaging system.

Early Stopping and Validation

Early stopping can be used to find the optimal number of boosting rounds. Set `early_stopping_rounds` to an integer value to monitor the performance of the model on the validation set while training. For information on how to configure the validation set, see the *Validation* section of `mlr3::Learner`. The internal validation measure can be set the `eval` parameter which should be a list of mlr3::Measures, functions, or strings for the internal lightgbm measures. If `first_metric_only = FALSE` (default), the learner stops when any metric fails to improve.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrLightGBM`

Active bindings

`internal_valid_scores` The validation scores for all metrics at the best iteration (`model$best_iter`), which is also the iteration that LightGBM predicts with. Extracted from `model$record_evals`.

`best_valid_scores` The validation scores for all metrics at the best iteration (`model$best_iter`), i.e. the iteration that is also reported via `$internal_tuned_values`. Because LightGBM also predicts with the best iteration, these are identical to `$internal_valid_scores` whenever early stopping is activated. If early stopping is not activated, no best iteration is tracked and this is an empty list.

`internal_tuned_values` Returns the early stopped iterations if `early_stopping_rounds` was set during training.

`validate` How to construct the internal validation data. This parameter can be either `NULL`, a ratio, `"test"`, or `"predefined"`.

Methods

Public methods:

- `LearnerRegrLightGBM$new()`
- `LearnerRegrLightGBM$importance()`
- `LearnerRegrLightGBM$clone()`

`LearnerRegrLightGBM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrLightGBM$new()
```

`LearnerRegrLightGBM$importance()`: The importance scores are extracted from `lbg.importance`.

Usage:

```
LearnerRegrLightGBM$importance()
```

Returns: Named `numeric()`.

`LearnerRegrLightGBM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrLightGBM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

kapsner

References

Ke, Guolin, Meng, Qi, Finley, Thomas, Wang, Taifeng, Chen, Wei, Ma, Weidong, Ye, Qiwei, Liu, Tie-Yan (2017). "Lightgbm: A highly efficient gradient boosting decision tree." *Advances in neural information processing systems*, **30**.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.lightgbm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())
```

```
# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.linear_regression`
Regression Linear Regression Learner From Weka

Description

Linear Regression learner that uses the Akaike criterion for model selection and is able to deal with weighted instances. Calls `RWeka::LinearRegression()` **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- `additional_stats`:
  - original id: additional-stats
- `use_qr`:
  - original id: use-qr
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.linear_regression")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
S	character	0	0, 1, 2	-
C	logical	FALSE	TRUE, FALSE	-
R	numeric	1e-08		$(-\infty, \infty)$
minimal	logical	FALSE	TRUE, FALSE	-
additional_stats	logical	FALSE	TRUE, FALSE	-
use_qr	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrLinearRegression`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrLinearRegression$new()`
- `LearnerRegrLinearRegression$marshal()`
- `LearnerRegrLinearRegression$unmarshal()`
- `LearnerRegrLinearRegression$clone()`

`LearnerRegrLinearRegression$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrLinearRegression$new()
```

`LearnerRegrLinearRegression$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrLinearRegression$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrLinearRegression$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrLinearRegression$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrLinearRegression$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrLinearRegression$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.linear_regression")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.lmer`

Regression Linear Mixed Effects Learner

Description

Linear model with random effects. Calls `lme4::lmer()` from **lme4**.

Formula

Although most mlr3 learners don't allow to specify the formula manually, and automatically set it by calling `task$formula()`, this learner allows to set the `formula` because it's core functionality depends it. This means that it might not always use all features that are available in the task. Be aware, that this can sometimes lead to unexpected error messages, because mlr3 checks the compatibility between the learner and the task on **all** available features.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.lmer")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **lme4**

Parameters

Id	Type	Default	Levels
formula	untyped	-	
REML	logical	TRUE	TRUE, FA
start	untyped	NULL	
verbose	integer	0	

contrasts	untyped	NULL	
optimizer	character	nloptwrap	Nelder_Me
restart_edge	logical	FALSE	TRUE, FA
boundary.tol	numeric	1e-05	
calc.derivs	logical	TRUE	TRUE, FA
autoscale	untyped	NULL	
check.nobs.vs.rankZ	character	ignore	ignore, wa
check.nobs.vs.nlev	character	stop	ignore, wa
check.nlev.gtreq.5	character	ignore	ignore, wa
check.nlev.gtr.1	character	stop	ignore, wa
check.nobs.vs.nRE	character	stop	ignore, wa
check.rankX	character	message+drop.cols	message+d
check.scaleX	character	warning	warning, st
check.formula.LHS	character	stop	ignore, wa
check.conv.nobsmax	numeric	10000	
check.conv.nparmax	numeric	10	
check.conv.grad	untyped	"lme4::.makeCC(\"warning\", tol = 2e-3, relTol = NULL)"	
check.conv.singular	untyped	"lme4::.makeCC(action = \"message\", tol = formals(lme4::isSingular)$tol)"	
check.conv.hess	untyped	"lme4::.makeCC(action = \"warning\", tol = 1e-6)"	
optCtrl	untyped	list()	
newparams	untyped	NULL	
re.form	untyped	NULL	
random.only	logical	FALSE	TRUE, FA
allow.new.levels	logical	FALSE	TRUE, FA
na.action	untyped	"stats::na.pass"	

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated during training via the `offset` argument in `lme4::lmer()`. No offset is applied during prediction for this learner.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrLmer`

Methods

Public methods:

- `LearnerRegrLmer$new()`
- `LearnerRegrLmer$clone()`

`LearnerRegrLmer$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrLmer$new()
```

`LearnerRegrLmer$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrLmer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

s-kganz

References

Bates, M D (2010). "lme4: Mixed-effects modeling with R."

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner and set parameter values
learner = lrn("regr.lmer", formula = mpg ~ (1 | gear))

# Define a Task
task = tsk("mtcars")

# Train the learner
learner$train(task)

print(learner$model)
```

`mlr_learners_regr.m5p` *Regression M5P Learner*

Description

Implements base routines for generating M5 Model trees and rules. Calls `RWeka::M5P()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.m5p")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
N	logical	-	TRUE, FALSE	-
U	logical	-	TRUE, FALSE	-
R	logical	-	TRUE, FALSE	-
M	integer	4		$(-\infty, \infty)$

output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
L	logical	-	TRUE, FALSE	-
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrM5P`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrM5P$new()`
- `LearnerRegrM5P$marshal()`
- `LearnerRegrM5P$unmarshal()`
- `LearnerRegrM5P$clone()`

`LearnerRegrM5P$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrM5P$new()
```

`LearnerRegrM5P$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrM5P$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrM5P$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrM5P$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrM5P$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrM5P$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Quinlan RJ (1992). "Learning with Continuous Classes." In *5th Australian Joint Conference on Artificial Intelligence*, 343-348.

Wang Y, Witten IH (1997). "Induction of model trees for predicting continuous classes." In *Poster papers of the 9th European Conference on Machine Learning*.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.m5p")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
```

```
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.M5Rules`
Regression M5Rules Learner

Description

Algorithm for inducing decision lists from model trees. Calls `RWeka::M5Rules()` from **RWeka**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.M5Rules")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
N	logical	FALSE	TRUE, FALSE	-
U	logical	FALSE	TRUE, FALSE	-
R	logical	FALSE	TRUE, FALSE	-
M	integer	4		$(-\infty, \infty)$
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrM5Rules`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrM5Rules$new()`
- `LearnerRegrM5Rules$marshal()`
- `LearnerRegrM5Rules$unmarshal()`
- `LearnerRegrM5Rules$clone()`

`LearnerRegrM5Rules$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrM5Rules$new()
```

`LearnerRegrM5Rules$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrM5Rules$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrM5Rules$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrM5Rules$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrM5Rules$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrM5Rules$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

henrifnk

References

Holmes, Geoffrey, Hall, Mark, Prank, Eibe (1999). "Generating rule sets from model trees." In *Australasian joint conference on artificial intelligence*, 1–12. Springer.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.M5Rules")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)
```

```
print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.mars`

Regression Mars Learner

Description

Multivariate Adaptive Regression Splines. Calls `mda::mars()` from **mda**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.mars")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3extralearners**, **mda**

Parameters

Id	Type	Default	Levels	Range
degree	integer	1		$[1, \infty)$
nk	integer	-		$[1, \infty)$
penalty	numeric	2		$[0, \infty)$
thresh	numeric	0.001		$[0, \infty)$
prune	logical	TRUE	TRUE, FALSE	-
trace.mars	logical	FALSE	TRUE, FALSE	-
forward.step	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrMars`

Methods

Public methods:

- `LearnerRegrMars$new()`
- `LearnerRegrMars$clone()`

`LearnerRegrMars$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrMars$new()
```

`LearnerRegrMars$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrMars$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sumny

References

Friedman, H J (1991). "Multivariate adaptive regression splines." *The annals of statistics*, **19**(1), 1–67.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.mars")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.mob *Regression Model-based Recursive Partitioning Learner*

Description

Model-based recursive partitioning algorithm. Calls `partykit::mob()` from **mob**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.mob")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **partykit**, **sandwich**, **coin**

Parameters

Id	Type	Default	Levels	Range
rhs	untyped	-		-
fit	untyped	-		-
offset	untyped	-		-
cluster	untyped	-		-
alpha	numeric	0.05		$[0, 1]$
bonferroni	logical	TRUE	TRUE, FALSE	-
minsize	integer	-		$[1, \infty)$
minsplit	integer	-		$[1, \infty)$
minbucket	integer	-		$[1, \infty)$
maxdepth	integer	Inf		$[0, \infty)$
mtry	integer	Inf		$[0, \infty)$
trim	numeric	0.1		$[0, \infty)$
breakties	logical	FALSE	TRUE, FALSE	-
parm	untyped	-		-
dfsplit	integer	-		$[0, \infty)$
prune	untyped	-		-
restart	logical	TRUE	TRUE, FALSE	-
verbose	logical	FALSE	TRUE, FALSE	-
caseweights	logical	TRUE	TRUE, FALSE	-
ytype	character	vector	vector, matrix, data.frame	-
xtype	character	matrix	vector, matrix, data.frame	-
terminal	untyped	"object"		-
inner	untyped	"object"		-
model	logical	TRUE	TRUE, FALSE	-
numsplit	character	left	left, center	-
catsplit	character	binary	binary, multiway	-
vcov	character	opg	opg, info, sandwich	-
ordinal	character	chisq	chisq, max, L2	-
nrep	integer	10000		$[0, \infty)$
applyfun	untyped	-		-
cores	integer	NULL		$(-\infty, \infty)$
additional	untyped	-		-
predict_fun	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrMob`

Methods

Public methods:

- `LearnerRegrMob$new()`
- `LearnerRegrMob$clone()`

`LearnerRegrMob$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrMob$new()
```

`LearnerRegrMob$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrMob$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sumny

References

Hothorn T, Zeileis A (2015). "partykit: A Modular Toolkit for Recursive Partytioning in R." *Journal of Machine Learning Research*, **16**(118), 3905-3909. http://jmlr.org/papers/v16/hothorn15a.html.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. doi:10.1198/106186006x133933. https://doi.org/10.1198/106186006x133933.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
library(mlr3)
lm_ = function(y, x, start = NULL, weights = NULL, offset = NULL, ...) {
  lm(y ~ 1, ...)
}
learner = LearnerRegrMob$new()
learner$param_set$values$rhs = "."
learner$param_set$values$fit = lm_
learner$feature_types = c("logical", "integer", "numeric", "factor", "ordered")
```

```
predict_fun = function(object, newdata, task, .type) {
  preds = predict(object, newdata = newdata, type = "response", se.fit = TRUE)
  cbind(preds$fit, preds$se.fit)
}
learner$param_set$values$predict_fun = predict_fun
task = tsk("mtcars")
ids = partition(task)
learner$train(task, row_ids = ids$train)
learner$predict(task, row_ids = ids$test)
```

`mlr_learners_regr.mqgam` *Generalized Additive Quantile Regression Model for Multiple Quantiles*

Description

Quantile Regression with generalized additive models for fitting a learner on multiple quantiles simultaneously. Calls `qgam::mqgam()` from package **qgam**.

Form

For the `form` parameter, a gam formula specific to the Task is required (see example and `?mgcv::formula.gam`). If no formula is provided, a fallback formula using all features in the task is used that will make the Learner behave like Linear Quantile Regression. The features specified in the formula need to be the same as columns with col_roles "feature" in the task.

Quantile

The quantiles for the Learner, i.e. `qu` parameter from `qgam::mqgam()`, is set using the values specified in `learner$quantiles`. The response quantile can be specified using `learner$quantile_response`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.mqgam")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “se”, “quantiles”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **qgam**

Parameters

Id	Type	Default	Levels	Range
form	untyped	-		-
lsig	numeric	-		$(-\infty, \infty)$
err	numeric	-		$[0, 1]$
cluster	untyped	NULL		-
multicore	logical	-	TRUE, FALSE	-
ncores	numeric	-		$(-\infty, \infty)$
paropts	untyped	list()		-
link	untyped	"identity"		-
argGam	untyped	-		-
discrete	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrMQGam`

Methods

Public methods:

- `LearnerRegrMQGam$new()`
- `LearnerRegrMQGam$clone()`

`LearnerRegrMQGam$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrMQGam$new()
```

`LearnerRegrMQGam$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrMQGam$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

lona-k

References

Fasiolo, Matteo, Wood, N. S, Zaffran, Margaux, Nedellec, Raphael, Goude, Yannig (2017). "Fast Calibrated Additive Quantile Regression." *Journal of the American Statistical Association*, **116**, 1402–1412. doi:10.1080/01621459.2020.1725521.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

`mlr_learners_regr.multilayer_perceptron`

Regression MultilayerPerceptron Learner

Description

Regressor that uses backpropagation to learn a multi-layer perceptron. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern
- `G` removed:
  - GUI will be opened
- Reason for change: The parameter is removed because we don't want to launch GUI.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.multilayer_perceptron")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
L	numeric	0.3		$[0, 1]$
M	numeric	0.2		$[0, 1]$
N	integer	500		$[1, \infty)$
V	numeric	0		$[0, 100]$
S	integer	0		$[0, \infty)$
E	integer	20		$[1, \infty)$
A	logical	FALSE	TRUE, FALSE	-
B	logical	FALSE	TRUE, FALSE	-
H	untyped	"a"		-
C	logical	FALSE	TRUE, FALSE	-
I	logical	FALSE	TRUE, FALSE	-
R	logical	FALSE	TRUE, FALSE	-
D	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrMultilayerPerceptron`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrMultilayerPerceptron$new()`
- `LearnerRegrMultilayerPerceptron$marshal()`

- `LearnerRegrMultilayerPerceptron$unmarshal()`
- `LearnerRegrMultilayerPerceptron$clone()`

`LearnerRegrMultilayerPerceptron$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrMultilayerPerceptron$new()
```

`LearnerRegrMultilayerPerceptron$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrMultilayerPerceptron$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrMultilayerPerceptron$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrMultilayerPerceptron$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrMultilayerPerceptron$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrMultilayerPerceptron$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.multilayer_perceptron")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.pcr *Principal Component Regression Learner*

Description

Principal Component Regression model. Calls `pls::pcr()` from **pls**.

Initial parameter values

- `model`
  - Actual default: `TRUE`
  - Adjusted default: `FALSE`
  - Reason for change: keeps model objects compact because predictions do not require the stored model frame.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.pcr")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **pls**

Parameters

Id	Type	Default	Levels	Range
ncomp	integer	-		$[1, \infty)$
method	character	cppls	kernelpls, widekernelpls, simpls, oscorespls, cppls, svdpc	-
scale	logical	FALSE	TRUE, FALSE	-
model	logical	-	TRUE, FALSE	-
x	logical	FALSE	TRUE, FALSE	-
y	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrPcr`

Methods

Public methods:

- `LearnerRegrPcr$new()`
- `LearnerRegrPcr$clone()`

`LearnerRegrPcr$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrPcr$new()
```

`LearnerRegrPcr$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrPcr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

During prediction, all components `ncomp` used in training are used.

Author(s)

awinterstetter

References

Jolliffe, T. I (1982). "A note on the use of principal components in regression." *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **31**(3), 300–303.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.pcr")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.penalized`

Regression L1 and L2 Penalized Learner

Description

Penalized (L1 and L2) Logistic Regression. Calls `penalized::penalized()` from: **penalized**.

Details

The `penalized` and `unpenalized` arguments in the learner are implemented slightly differently than in `penalized::penalized()`. Here, there is no parameter for `penalized` but instead it is assumed that every variable is penalized unless stated in the `unpenalized` parameter.

Initial parameter values

- `trace` is set to `"FALSE"` to disable printing output during model training.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.penalized")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **penalized**

Parameters

Id	Type	Default	Levels	Range
epsilon	numeric	1e-10		$[0, \infty)$
fusedl	logical	FALSE	TRUE, FALSE	-
lambda1	numeric	0		$[0, \infty)$
lambda2	numeric	0		$[0, \infty)$
maxiter	integer	-		$[1, \infty)$
model	character	linear	linear, poisson	-
positive	untyped	FALSE		-
standardize	logical	FALSE	TRUE, FALSE	-
startbeta	untyped	-		-
startgamma	untyped	-		-
steps	untyped	1L		-
trace	logical	TRUE	TRUE, FALSE	-

unpenalized	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrPenalized`

Methods

Public methods:

- `LearnerRegrPenalized$new()`
- `LearnerRegrPenalized$clone()`

`LearnerRegrPenalized$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrPenalized$new()
```

`LearnerRegrPenalized$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrPenalized$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

annanzrv

References

Goeman, J J (2010). "L1 penalized estimation in the Cox proportional hazards model." *Biometrical journal*, **52**(1), 70–84.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.penalized")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.plsr`

Partial Least Squares Regression

Description

Partial least squares regression. Calls `pls::plsr()` from **pls**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.plsr")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **pls**

Parameters

Id	Type	Default	Levels	Range
ncomp	integer	-		$[1, \infty)$
method	character	kernelpls	kernelpls, widekernelpls, simpls, oscorespls	-
scale	logical	TRUE	TRUE, FALSE	-
center	logical	TRUE	TRUE, FALSE	-
validation	character	none	none, CV, LOO	-
model	logical	TRUE	TRUE, FALSE	-
x	logical	FALSE	TRUE, FALSE	-
y	logical	FALSE	TRUE, FALSE	-
orthScores	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrPlsr`

Methods

Public methods:

- `LearnerRegrPlsr$new()`
- `LearnerRegrPlsr$clone()`

`LearnerRegrPlsr$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrPlsr$new()
```

`LearnerRegrPlsr$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrPlsr$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

During prediction, all components `ncomp` used in training are used.

Author(s)

awinterstetter

References

Mevik, Bjorn-Helge, Wehrens, Ron (2007). "The pls Package: Principal Component and Partial Least Squares Regression in R." *Journal of Statistical Software*, **18**(2), 1–24. doi:10.18637/jss.v018.i02.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.plsr")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.polyFit`

Polynomial Regression Learner

Description

Polynomial regression. Calls `polyreg::polyFit()` from **polyreg**.

Initial parameter values

- `deg`: We have set this to 2, pretty arbitrarily.
- `noisy`: We have set this to FALSE, to get no output on the console.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.polyFit")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **polyreg**

Parameters

Id	Type	Default	Levels	Range
deg	integer	-		$[0, \infty)$
maxInteractDeg	integer	-		$[0, \infty)$
return_xy	logical	FALSE	TRUE, FALSE	-
returnPoly	logical	FALSE	TRUE, FALSE	-
noisy	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrPolyFit`

Methods

Public methods:

- `LearnerRegrPolyFit$new()`
- `LearnerRegrPolyFit$clone()`

`LearnerRegrPolyFit$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrPolyFit$new()
```

`LearnerRegrPolyFit$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrPolyFit$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

`mlr_learners_regr.polynomial`

Polynomial Regression Learner

Description

Polynomial regression without interactions between features. Calls `stats::lm()` from base package 'stats' on a model formula in which every feature is expanded with `stats::poly()`. For polynomial regression that also includes interaction terms, see `regr.polyFit`.

Initial parameter values

- `degree`
  - Actual default: `1`
  - Adjusted default: `2`
  - Reason for change: A degree of `1` is plain linear regression, so the lowest degree that makes this learner differ from `stats::lm()` is the more useful default.

Polynomial degree

`stats::poly()` requires the degree of the polynomial to be smaller than the number of distinct values of a feature. To keep the learner usable inside resampling, where individual folds may contain features with few distinct values, the degree is capped per feature at `min(degree, <number of distinct values> - 1)`. Features that are constant in the training data enter the formula untransformed.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.polynomial")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “se”
- Feature Types: “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3extralearners**, ’stats’

Parameters

Id	Type	Default	Levels	Range
degree	integer	1		$[1, \infty)$
raw	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrPolynomial`

Methods

Public methods:

- `LearnerRegrPolynomial$new()`
- `LearnerRegrPolynomial$clone()`

`LearnerRegrPolynomial$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrPolynomial$new()
```

`LearnerRegrPolynomial$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrPolynomial$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

imanechadli2003

References

Hastie, Trevor, Tibshirani, Robert, Friedman, Jerome (2009). *The Elements of Statistical Learning*, series Springer Series in Statistics, 2 edition. Springer, New York. doi:10.1007/9780387848587.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.polynomial")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.priority_lasso`

Regression Priority Lasso Learner

Description

Patient outcome prediction based on multi-omics data taking practitioners' preferences into account. Calls `prioritylasso::prioritylasso()` from **prioritylasso**.

Initial parameter values

- `family` is set to `"gaussian"` and cannot be changed
- `type.measure` set to `"mse"` (cross-validation measure) and cannot be changed

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.priority_lasso")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: **mlr3**, **prioritylasso**

Parameters

Id	Type	Default	Levels	Range
blocks	untyped	-		-
max.coef	untyped	NULL		-
block1.penalization	logical	TRUE	TRUE, FALSE	-
lambda.type	character	lambda.min	lambda.min, lambda.1se	-
standardize	logical	TRUE	TRUE, FALSE	-
nfolds	integer	10		$[3, \infty)$
foldid	untyped	NULL		-
cvoffset	logical	FALSE	TRUE, FALSE	-
cvoffsetnfolds	integer	10		$[1, \infty)$
scale.y	logical	FALSE	TRUE, FALSE	-
return.x	logical	TRUE	TRUE, FALSE	-
lambda	untyped	NULL		-
grouped	logical	TRUE	TRUE, FALSE	-
trace.it	integer	0		$[0, 1]$
type.gaussian	character	-	covariance, naive	-
include.allintercepts	logical	FALSE	TRUE, FALSE	-
use.blocks	untyped	"all"		-
adaptive.order	logical	FALSE	TRUE, FALSE	-

Scope and supported arguments

This learner intentionally exposes a focused subset of training and prediction arguments. It is designed to work well out of the box, without requiring extensive parameter tuning. Some arguments from `cv.glmnet()`, `glmnet()`, and `predict.prioritylasso()` are not included, because they are

not consistently supported or forwarded through the full train/predict path (e.g. handling missing test data, or performing relaxed lasso fits).

Please open an issue if there is a need for supporting more learner parameters.

Custom mlr3 parameters

- `adaptive.order`: if `TRUE`, the priority order of blocks is estimated from the data following Herrmann et al. (2021), instead of using the user-supplied block order. For each block, a Ridge regression (`alpha = 0`) is fit using `cv.glmnet()` on that block alone. The importance of a block is measured by the mean absolute coefficient (MAC) score at the `lambda.min` value (the lambda giving minimum cross-validation error). A penalty factor of `1 / MAC` is then assigned to each block. Blocks are sorted by increasing penalty factor, i.e., blocks with larger MAC (stronger average signal) receive higher priority (come first). Also, the block-wise penalty factors are attached to the fitted model object as `learner$model$block.penalty.factors`.

This method is useful when no domain knowledge is available to specify block priority. In this step, data are **standardized by default** (`standardize = TRUE`), but this can be overridden by the learner's `standardize` parameter. `lambda.min` is always used to derive the block priority. Additional arguments such as `nfolds`, `type.measure`, `weights` and `cox.ties` (if provided) are forwarded to each block-wise `cv.glmnet()` fit. The `max.coef` parameter, if supplied, it is re-ordered accordingly to align with the new block order.

This parameter is ignored when fewer than two blocks are provided. It defaults to `FALSE` for backward compatibility.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrPriorityLasso`

Methods

Public methods:

- `LearnerRegrPriorityLasso$new()`
- `LearnerRegrPriorityLasso$selected_features()`
- `LearnerRegrPriorityLasso$clone()`

`LearnerRegrPriorityLasso$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrPriorityLasso$new()
```

`LearnerRegrPriorityLasso$selected_features()`: Selected features when coef is positive

Usage:

```
LearnerRegrPriorityLasso$selected_features()
```

Returns: `character()`.

`LearnerRegrPriorityLasso$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrPriorityLasso$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

HarutyunyanLiana

bblodfon

References

Simon K, Vindi J, Roman H, Tobias H, Anne-Laure B (2018). "Priority-Lasso: a simple hierarchical approach to the prediction of clinical outcome using multi-omics data." *BMC Bioinformatics*, **19**. doi:10.1186/s1285901823446.

Herrmann, M., Probst, P., Hornung, R., Jurinovic, V., Boulesteix, L. A (2021). "Large-scale benchmark study of survival prediction methods using multi-omics data." *Briefings in Bioinformatics*, **22**(3), 1–15. doi:10.1093/BIB/BBAA167.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Simulate regression data
set.seed(1L)
x = matrix(rnorm(50L * 500L), nrow = 50L, ncol = 500L)
data = as.data.frame(x)
data$y = rnorm(50L)

# Define a Task
task = mlr3::as_task_regr(data, target = "y")

# Create train and test set
ids = partition(task)

# Define the Learner and set parameter values
learner = lrn("regr.priority_lasso",
  blocks = list(bp1 = 1:75, bp2 = 76:200, bp3 = 201:500),
  max.coef = c(Inf, 8, 5),
  block1.penalization = TRUE,
  lambda.type = "lambda.min",
  standardize = TRUE,
  nfolds = 5,
```

```
  cvoffset = FALSE)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Selected features
learner$selected_features()

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.qgam`

Generalized Additive Quantile Regression Model

Description

Quantile Regression with generalized additive models. Calls `qgam::qgam()` from package **qgam**.

Form

For the `form` parameter, a gam formula specific to the Task is required (see example and `?mgcv::formula.gam`). If no formula is provided, a fallback formula using all features in the task is used that will make the Learner behave like Linear Quantile Regression. The features specified in the formula need to be the same as columns with col_roles "feature" in the task.

Quantile

The quantile for the Learner, i.e. `qu` parameter from `qgam::qgam()`, is set using the value specified in `learner$quantiles`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.qgam")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “se”, “quantiles”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **qgam**

Parameters

Id	Type	Default	Levels	Range
form	untyped	-		-
lsig	numeric	-		$(-\infty, \infty)$
err	numeric	-		$[0, 1]$
cluster	untyped	NULL		-
multicore	logical	-	TRUE, FALSE	-
ncores	numeric	-		$(-\infty, \infty)$
paropts	untyped	list()		-
link	untyped	"identity"		-
argGam	untyped	-		-
discrete	logical	FALSE	TRUE, FALSE	-
block.size	integer	1000		$(-\infty, \infty)$
unconditional	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrQGam`

Methods

Public methods:

- `LearnerRegrQGam$new()`
- `LearnerRegrQGam$clone()`

`LearnerRegrQGam$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrQGam$new()
```

`LearnerRegrQGam$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrQGam$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

lona-k

References

Fasiolo, Matteo, Wood, N. S, Zaffran, Margaux, Nedellec, Raphael, Goude, Yannig (2017). "Fast Calibrated Additive Quantile Regression." *Journal of the American Statistical Association*, **116**, 1402–1412. doi:10.1080/01621459.2020.1725521.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

`mlr_learners_regr.random_forest_weka`

Regression Random Forest Learner from Weka

Description

Class for constructing a forest of random trees. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- `store_out_of_bag_predictions`:
  - original id: store-out-of-bag-predictions
- `output_out_of_bag_complexity_statistics`:
  - original id: output-out-of-bag-complexity-statistics
- `num_slots`:
  - original id: num-slots
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern
- `attribute-importance` removed:
  - Compute and output attribute importance (mean impurity decrease method)
- Reason for change: The parameter is removed because it's unclear how to actually use it.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.random_forest_weka")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
P	numeric	100		$[0, 100]$
O	logical	FALSE	TRUE, FALSE	-
store_out_of_bag_predictions	logical	FALSE	TRUE, FALSE	-
output_out_of_bag_complexity_statistics	logical	FALSE	TRUE, FALSE	-
print	logical	FALSE	TRUE, FALSE	-
I	integer	100		$[1, \infty)$
num_slots	integer	1		$(-\infty, \infty)$
K	integer	0		$(-\infty, \infty)$
M	integer	1		$[1, \infty)$
V	numeric	0.001		$(-\infty, \infty)$
S	integer	1		$(-\infty, \infty)$
depth	integer	0		$[0, \infty)$
N	integer	0		$(-\infty, \infty)$
U	logical	FALSE	TRUE, FALSE	-
B	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrRandomForestWeka`

Active bindings

`marshaled (logical(1))`
: Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrRandomForestWeka$new()`
- `LearnerRegrRandomForestWeka$marshal()`
- `LearnerRegrRandomForestWeka$unmarshal()`
- `LearnerRegrRandomForestWeka$clone()`

`LearnerRegrRandomForestWeka$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrRandomForestWeka$new()
```

`LearnerRegrRandomForestWeka$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrRandomForestWeka$marshal(...)
```

Arguments:

`...` (any)
: Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrRandomForestWeka$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrRandomForestWeka$unmarshal(...)
```

Arguments:

`...` (any)
: Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrRandomForestWeka$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrRandomForestWeka$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Breiman, Leo (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565. doi:10.1023/A:1010933404324.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.random_forest_weka")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.random_tree`

Regression Random Tree Learner

Description

Tree that considers K randomly chosen attributes at each node. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.random_tree")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
K	integer	0		$[0, \infty)$
M	integer	1		$[1, \infty)$
V	numeric	0.001		$(-\infty, \infty)$
S	integer	1		$(-\infty, \infty)$
depth	integer	0		$[0, \infty)$
N	integer	0		$[0, \infty)$
U	logical	FALSE	TRUE, FALSE	-
B	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrRandomTree`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrRandomTree$new()`
- `LearnerRegrRandomTree$marshal()`
- `LearnerRegrRandomTree$unmarshal()`
- `LearnerRegrRandomTree$clone()`

`LearnerRegrRandomTree$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrRandomTree$new()
```

`LearnerRegrRandomTree$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrRandomTree$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrRandomTree$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrRandomTree$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrRandomTree$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrRandomTree$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.random_tree")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.randomForest`

Regression Random Forest Learner

Description

Random forest for regression. Calls `randomForest::randomForest()` from **randomForest**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.randomForest")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3extralearners**, **randomForest**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
mtry	integer	-		$[1, \infty)$
replace	logical	TRUE	TRUE, FALSE	-
strata	untyped	-		-
sampsize	untyped	-		-
nodesize	integer	5		$[1, \infty)$
maxnodes	integer	-		$[1, \infty)$
importance	character	FALSE	mse, nudepurity, none	-
localImp	logical	FALSE	TRUE, FALSE	-
proximity	logical	FALSE	TRUE, FALSE	-
oob.prox	logical	-	TRUE, FALSE	-
norm.votes	logical	TRUE	TRUE, FALSE	-
do.trace	logical	FALSE	TRUE, FALSE	-
keep.forest	logical	TRUE	TRUE, FALSE	-
keep.inbag	logical	FALSE	TRUE, FALSE	-
predict.all	logical	FALSE	TRUE, FALSE	-
nodes	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrRandomForest`

Methods

Public methods:

- `LearnerRegrRandomForest$new()`
- `LearnerRegrRandomForest$importance()`
- `LearnerRegrRandomForest$oob_error()`

- `LearnerRegrRandomForest$clone()`

`LearnerRegrRandomForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrRandomForest$new()
```

`LearnerRegrRandomForest$importance()`: The importance scores are extracted from the slot `importance`. Parameter 'importance' must be set to either `"mse"` or `"nodepurity"`.

Usage:

```
LearnerRegrRandomForest$importance()
```

Returns: Named `numeric()`.

`LearnerRegrRandomForest$oob_error()`: OOB errors are extracted from the model slot `mse`.

Usage:

```
LearnerRegrRandomForest$oob_error()
```

Returns: `numeric(1)`.

`LearnerRegrRandomForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrRandomForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

pat-s

References

Breiman, Leo (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565. doi:10.1023/A:1010933404324.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.randomForest", importance = "mse")
print(learner)

# Define a Task
task = tsk("mtcars")
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.reptree`

Regression Decision Tree Learner

Description

Fast decision tree learner. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.reptree")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
M	integer	2		$(-\infty, \infty)$
V	numeric	0.001		$(-\infty, \infty)$
N	integer	3		$(-\infty, \infty)$
S	integer	1		$(-\infty, \infty)$
P	logical	-	TRUE, FALSE	-
L	integer	-1		$(-\infty, \infty)$
I	integer	0		$(-\infty, \infty)$
R	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrREPTree`

Active bindings

`marshaled (logical(1))`
: Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrREPTree$new()`
- `LearnerRegrREPTree$marshal()`
- `LearnerRegrREPTree$unmarshal()`
- `LearnerRegrREPTree$clone()`

`LearnerRegrREPTree$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrREPTree$new()
```

`LearnerRegrREPTree$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrREPTree$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrREPTree$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrREPTree$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrREPTree$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrREPTree$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.reptree")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.rfsrc`

Regression Random Forest SRC Learner

Description

Random forest for regression. Calls `randomForestSRC::rfsrc()` from **randomForestSRC**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.rfsrc")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **randomForestSRC**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
mtry	integer	-		$[1, \infty)$
mtry.ratio	numeric	-		$[0, 1]$
nodesize	integer	15		$[1, \infty)$
nodedepth	integer	-		$[1, \infty)$
splitrule	character	mse	mse, quantile.regr, la.quantile.regr	-
nsplit	integer	10		$[0, \infty)$
importance	character	FALSE	FALSE, TRUE, none, permute, random, anti	-
block.size	integer	10		$[1, \infty)$
bootstrap	character	by.root	by.root, by.node, none, by.user	-
samptype	character	swor	swor, swr	-
samp	untyped	-		-
membership	logical	FALSE	TRUE, FALSE	-
sampsize	untyped	-		-
sampsize.ratio	numeric	-		$[0, 1]$
na.action	character	na.omit	na.omit, na.impute	-
nimpute	integer	1		$[1, \infty)$
proximity	character	FALSE	FALSE, TRUE, inbag, oob, all	-
distance	character	FALSE	FALSE, TRUE, inbag, oob, all	-
forest.wt	character	FALSE	FALSE, TRUE, inbag, oob, all	-
xvar.wt	untyped	-		-
split.wt	untyped	-		-
forest	logical	TRUE	TRUE, FALSE	-
var.used	character	FALSE	FALSE, all.trees	-
split.depth	character	FALSE	FALSE, all.trees, by.tree	-
seed	integer	-		$(-\infty, -1]$
do.trace	logical	FALSE	TRUE, FALSE	-
get.tree	untyped	-		-
outcome	character	train	train, test	-
ptn.count	integer	0		$[0, \infty)$
cores	integer	1		$[1, \infty)$
save.memory	logical	FALSE	TRUE, FALSE	-
perf.type	character	-	none	-
case.depth	logical	FALSE	TRUE, FALSE	-
marginal.xvar	untyped	NULL		-

Custom mlr3 parameters

- `mtry`: This hyperparameter can alternatively be set via the added hyperparameter `mtry.ratio` as `mtry = max(ceiling(mtry.ratio * n_features), 1)`. Note that `mtry` and `mtry.ratio` are mutually exclusive.

- `sampsize`: This hyperparameter can alternatively be set via the added hyperparameter `sampsize.ratio` as `sampsize = max(ceiling(sampsize.ratio * n_obs), 1)`. Note that `sampsize` and `sampsize.ratio` are mutually exclusive.
- `cores`: This value is set as the option `rf.cores` during training and is set to 1 by default.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrRandomForestSRC`

Methods

Public methods:

- `LearnerRegrRandomForestSRC$new()`
- `LearnerRegrRandomForestSRC$importance()`
- `LearnerRegrRandomForestSRC$selected_features()`
- `LearnerRegrRandomForestSRC$oob_error()`
- `LearnerRegrRandomForestSRC$clone()`

`LearnerRegrRandomForestSRC$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrRandomForestSRC$new()
```

`LearnerRegrRandomForestSRC$importance()`: The importance scores are extracted from the model slot `importance`.

Usage:

```
LearnerRegrRandomForestSRC$importance()
```

Returns: Named `numeric()`.

`LearnerRegrRandomForestSRC$selected_features()`: Selected features are extracted from the model slot `var.used`.

Note: Due to a known issue in `randomForestSRC`, enabling `var.used = "all.trees"` causes prediction to fail. Therefore, this setting should be used exclusively for feature selection purposes and not when prediction is required.

Usage:

```
LearnerRegrRandomForestSRC$selected_features()
```

Returns: `character()`.

`LearnerRegrRandomForestSRC$oob_error()`: OOB error extracted from the model slot `err.rate`.

Usage:

```
LearnerRegrRandomForestSRC$oob_error()
```

Returns: `numeric()`.

`LearnerRegrRandomForestSRC$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrRandomForestSRC$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

RaphaelS1

References

Breiman, Leo (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565. doi:10.1023/A:1010933404324.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.rfsrc", importance = "TRUE")
print(learner)

# Define a Task
task = tsk("mtcars")
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.RRF` *Regression Regularized Random Forest Learner*

Description

Regularized random forest for regression. Calls `RRF::RRF()` from **RRF**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.RRF")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3extralearners**, **RRF**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
mtry	integer	-		$[1, \infty)$
nodesize	integer	-		$[1, \infty)$
replace	logical	TRUE	TRUE, FALSE	-
flagReg	integer	1		$[0, \infty)$
coefReg	numeric	0.8		$(-\infty, \infty)$
feaIni	untyped	-		-
corr.bias	logical	FALSE	TRUE, FALSE	-
maxnodes	integer	-		$[1, \infty)$
importance	logical	FALSE	TRUE, FALSE	-
localImp	logical	FALSE	TRUE, FALSE	-
nPerm	integer	1		$[1, \infty)$
proximity	logical	FALSE	TRUE, FALSE	-
oob.prox	logical	FALSE	TRUE, FALSE	-
do.trace	logical	FALSE	TRUE, FALSE	-
keep.inbag	logical	FALSE	TRUE, FALSE	-
keep.forest	logical	TRUE	TRUE, FALSE	-
strata	untyped	-		-
sampsize	untyped	-		-
predict.all	logical	FALSE	TRUE, FALSE	-
nodes	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrRRF`

Methods

Public methods:

- `LearnerRegrRRF$new()`
- `LearnerRegrRRF$importance()`
- `LearnerRegrRRF$oob_error()`
- `LearnerRegrRRF$clone()`

`LearnerRegrRRF$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrRRF$new()
```

`LearnerRegrRRF$importance()`: The importance scores are extracted from the slot `importance`.

Usage:

```
LearnerRegrRRF$importance()
```

Returns: Named `numeric()`.

`LearnerRegrRRF$oob_error()`: OOB errors are extracted from the model slot `mse`.

Usage:

```
LearnerRegrRRF$oob_error()
```

Returns: `numeric(1)`.

`LearnerRegrRRF$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrRRF$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

awinterstetter

References

Deng, Houtao, Runger, George (2012). "Feature selection via regularized trees." In *2012 International Joint Conference on Neural Networks (IJCNN)*, 1–8. IEEE. doi:10.1109/IJCNN.2012.6252640.

Deng, Houtao, Runger, George (2013). "Gene selection with guided regularized random forest." *Pattern Recognition*, **46**(12), 3483–3489. doi:10.1016/j.patcog.2013.05.021.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.RRF")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.rsm` *Regression Response Surface Model Learner*

Description

Fit a linear model with a response-surface component. Calls `rsm::rsm()` from **rsm**.

Custom mlr3 parameters

- `modelfun`: This parameter controls how the formula for `rsm::rsm()` is created. Possible values are:
  - `"FO"` - first order
  - `"TWI"` - wo-way interactions, this is with 1st oder terms
  - `"SO"` - full second order

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.rsm")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **rsm**

Parameters

Id	Type	Default	Levels
modelfun	character	-	FO, TWI, SO

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrRSM`

Methods

Public methods:

- `LearnerRegrRSM$new()`
- `LearnerRegrRSM$clone()`

`LearnerRegrRSM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrRSM$new()
```

`LearnerRegrRSM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrRSM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

sebffischer

References

Lenth, V R (2010). "Response-surface methods in R, using rsm." *Journal of Statistical Software*, **32**, 1–17.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.rsm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.rvm` *Regression Relevance Vector Machine Learner*

Description

Bayesian version of the support vector machine. Parameters `sigma`, `degree`, `scale`, `offset`, `order`, `length`, `lambda`, and `normalized` are added to make tuning `kpar` easier. If `kpar` is provided then these new parameters are ignored. If none are provided then the default "automatic" is used for `kpar`. Calls `kernlab::rvm()` from package **kernlab**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.rvm")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3extralearners**, **kernlab**

Parameters

Id	Type	Default	Levels
kernel	character	rbfdot	rbfdot, polydot, vanilladot, tanhdot, laplacedot, besseldot, anovadot, splinedot, stri
sigma	numeric	-	
degree	numeric	-	
scale	numeric	-	
offset	numeric	-	
order	numeric	-	
length	integer	-	
lambda	numeric	-	
normalized	logical	-	TRUE, FALSE
kpar	untyped	"automatic"	
alpha	untyped	5	
var	numeric	0.1	
var.fix	logical	FALSE	TRUE, FALSE
iterations	integer	100	
tol	numeric	2.220446e-16	
minmaxdiff	numeric	0.001	
verbosity	logical	FALSE	TRUE, FALSE
fit	logical	TRUE	TRUE, FALSE
na.action	untyped	na.omit	

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrRVM`

Methods

Public methods:

- `LearnerRegrRVM$new()`
- `LearnerRegrRVM$clone()`

`LearnerRegrRVM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrRVM$new()
```

`LearnerRegrRVM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrRVM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Karatzoglou, Alexandros, Smola, Alex, Hornik, Kurt, Zeileis, Achim (2004). "kernlab-an S4 package for kernel methods in R." *Journal of statistical software*, **11**(9), 1–20.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.rvm")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.sgd *Regression Stochastic Gradient Descent Learner*

Description

Stochastic Gradient Descent for learning various linear models. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Initial parameter values

- `F`:
  - Has only 3 out of 5 original loss functions: 2 = squared loss (regression), 3 = epsilon insensitive loss (regression) and 4 = Huber loss (regression) with 2 (squared loss) being the new default
  - Reason for change: this learner should only contain loss functions appropriate for regression tasks

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities

- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.sgd")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
F	character	2	2, 3, 4	-
L	numeric	0.01		$(-\infty, \infty)$
R	numeric	1e-04		$(-\infty, \infty)$
E	integer	500		$(-\infty, \infty)$
C	numeric	0.001		$(-\infty, \infty)$
N	logical	-	TRUE, FALSE	-
M	logical	-	TRUE, FALSE	-
S	integer	1		$(-\infty, \infty)$
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrSGD`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrSGD$new()`
- `LearnerRegrSGD$marshal()`
- `LearnerRegrSGD$unmarshal()`
- `LearnerRegrSGD$clone()`

`LearnerRegrSGD$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrSGD$new()
```

`LearnerRegrSGD$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrSGD$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrSGD$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrSGD$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrSGD$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrSGD$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.sgd")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_regr.simple_linear_regression

Regression SimpleLinearRegression Learner from Weka

Description

Simple linear regression model that picks the attribute that results in the lowest squared error. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- `additional_stats`:
  - original id: additional-stats
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.simple_linear_regression")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels	Range
subset	untyped	-		-
na.action	untyped	-		-
additional_stats	logical	FALSE	TRUE, FALSE	-
output_debug_info	logical	FALSE	TRUE, FALSE	-
do_not_check_capabilities	logical	FALSE	TRUE, FALSE	-
num_decimal_places	integer	2		$[1, \infty)$
batch_size	integer	100		$[1, \infty)$
options	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrSimpleLinearRegression`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrSimpleLinearRegression$new()`
- `LearnerRegrSimpleLinearRegression$marshal()`
- `LearnerRegrSimpleLinearRegression$unmarshal()`
- `LearnerRegrSimpleLinearRegression$clone()`

`LearnerRegrSimpleLinearRegression$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrSimpleLinearRegression$new()
```

`LearnerRegrSimpleLinearRegression$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrSimpleLinearRegression$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrSimpleLinearRegression$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrSimpleLinearRegression$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrSimpleLinearRegression$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrSimpleLinearRegression$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.simple_linear_regression")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.smo_reg`

Regression Support Vector Machine Learner

Description

Support Vector Machine for regression. Calls `RWeka::make_Weka_classifier()` from **RWeka**.

Custom mlr3 parameters

- `output_debug_info`:
  - original id: output-debug-info
- `do_not_check_capabilities`:
  - original id: do-not-check-capabilities
- `num_decimal_places`:
  - original id: num-decimal-places
- `batch_size`:
  - original id: batch-size
- `T_improved`:
  - original id: T
- `V_improved`:
  - original id: V
- `P_improved`:
  - original id: P
- `L_improved`:
  - original id: L (duplicated L for when I is set to RegSMOImproved)
- `W_improved`:
  - original id: W
- `C_poly`:
  - original id: C
- `E_poly`:
  - original id: E
- `L_poly`:
  - original id: L (duplicated L for when K is set to PolyKernel)
- Reason for change: This learner contains changed ids of the following control arguments since their ids contain irregular pattern

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.smo_reg")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **RWeka**

Parameters

Id	Type	Default	Levels
subset	untyped	-	
na.action	untyped	-	
C	numeric	1	
N	character	0	0, 1, 2
I	character	RegSMOImproved	RegSMO, RegSMOImproved
K	character	PolyKernel	NormalizedPolyKernel, PolyKernel, Puk, RBFKernel, StringK
T_improved	numeric	0.001	
V_improved	logical	TRUE	TRUE, FALSE
P_improved	numeric	1e-12	
L_improved	numeric	0.001	
W_improved	integer	1	
C_poly	integer	250007	
E_poly	numeric	1	
L_poly	logical	FALSE	TRUE, FALSE
output_debug_info	logical	FALSE	TRUE, FALSE
do_not_check_capabilities	logical	FALSE	TRUE, FALSE
num_decimal_places	integer	2	
batch_size	integer	100	
options	untyped	NULL	

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrSMOreg`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrSMOreg$new()`
- `LearnerRegrSMOreg$marshal()`
- `LearnerRegrSMOreg$unmarshal()`
- `LearnerRegrSMOreg$clone()`

`LearnerRegrSMOreg$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrSMOreg$new()
```

`LearnerRegrSMOreg$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrSMOreg$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrSMOreg$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrSMOreg$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrSMOreg$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrSMOreg$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

damirpolat

References

Shevade S, Keerthi S, Bhattacharyya C, Murthy K (1999). "Improvements to the SMO Algorithm for SVM Regression." In *IEEE Transactions on Neural Networks*.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("regr.smo_reg")
print(learner)

# Define a Task
task = tsk("mtcars")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_regr.tabfm`

TabFM Regression Learner

Description

Tabular foundation model developed by Google Research. Uses **reticulate** to interface with the `tabfm` Python package. The model makes zero-shot predictions via in-context learning and is not trained on the data in the usual sense. Features are passed to `tabfm` as a data frame, so that categorical features are ordinal encoded and missing values are imputed by the package's own preprocessing pipeline.

Installation

This learner relies on **reticulate** to handle Python dependencies. It is not necessary to install any Python package manually in advance or specify a Python environment via `reticulate::use_python()`, `reticulate::use_virtualenv()`, `reticulate::use_condaenv()`, or `reticulate::use_miniconda()`. By calling `$train()` or `$predict()`, the required Python packages will be installed automatically, if not already. The backend parameter selects which extra is installed: `"jax"` installs `tabfm[jax]` and `"pytorch"` installs `tabfm[pytorch]`. Reticulate will then configure and initialize an ephemeral environment satisfying those requirements, unless an existing environment (e.g., `"r-reticulate"`) in reticulate's Order of Discovery contains all the necessary packages.

You may also manually install `tabfm` into a Python environment following the official installation guide and specify the environment via `reticulate::use_*()` before calling `$train()` or `$predict()`.

On the first call to `$train()`, the pre-trained model weights (about 100 MB) are downloaded from Hugging Face and cached for subsequent calls.

Saving a Learner

In order to save a `LearnerRegrTabFM` for later usage, it is necessary to call the `$marshal()` method on the `Learner` before writing it to disk, as the object will otherwise not be saved correctly. After loading a marshaled `LearnerRegrTabFM` into R again, you then need to call `$unmarshal()` to transform it into a useable state. The several gigabyte pre-trained backbone is not serialized; it is detached before marshaling and reloaded from the Hugging Face cache on `$unmarshal()`.

Custom mlr3 parameters

- `backend` selects the computational backend, either `"jax"` or `"pytorch"`. It also determines which pre-trained weights are downloaded and which Python packages are installed.
- `device` is only used by the `"pytorch"` backend and is passed as a string to the model loader, e.g., `"cpu"` or `"cuda"`. The `"jax"` backend ignores this parameter and selects the device automatically.
- `norm_methods` accepts a single string or a character vector with values from `"none"`, `"power"`, `"quantile"`, `"quantile_rtdl"`, and `"robust"`.
- `n_feature_crosses` and `n_svd_features` accept either a non-negative integer or the string `"sqrt"`.
- `cache_context` is only supported by the `"pytorch"` backend.
- `random_state` accepts either an integer or the special value `"None"` which corresponds to `None` in Python.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.tabfm")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **reticulate**

Parameters

Id	Type	Default	Levels	Range
backend	character	-	jax, pytorch	-
device	untyped	-		-
n_estimators	integer	32		$[1, \infty)$
norm_methods	untyped	-		-

feat_shuffle_method	character	random	random, none	-
permute_categorical	logical	FALSE	TRUE, FALSE	-
outlier_threshold	numeric	4		$(-\infty, \infty)$
max_num_features	integer	500		$[1, \infty)$
max_num_rows	integer	-		$[1, \infty)$
use_amp	logical	TRUE	TRUE, FALSE	-
batch_size	integer	1		$[1, \infty)$
random_state	integer	42		$(-\infty, \infty)$
verbose	logical	FALSE	TRUE, FALSE	-
cat_encoder_mode	character	appearance	appearance, frequency	-
num_folds_for_cv	integer	5		$[2, \infty)$
n_feature_crosses	untyped	0L		-
n_svd_features	untyped	0L		-
total_svd_pool	integer	-		$[1, \infty)$
enable_nnls	logical	FALSE	TRUE, FALSE	-
nnls_beta	numeric	0.75		$(-\infty, \infty)$
min_rows_for_single_val_split	integer	2000		$[0, \infty)$
cache_context	logical	FALSE	TRUE, FALSE	-
maybe_quantize_kv_cache	logical	TRUE	TRUE, FALSE	-
keep_cache_on_device	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrTabFM`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrTabFM$new()`
- `LearnerRegrTabFM$marshal()`
- `LearnerRegrTabFM$unmarshal()`
- `LearnerRegrTabFM$clone()`

`LearnerRegrTabFM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrTabFM$new()
```

`LearnerRegrTabFM$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrTabFM$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrTabFM$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrTabFM$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrTabFM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrTabFM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Google Research (2026). "TabFM: Tabular Foundation Models." https://github.com/google-research/tabfm.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

`mlr_learners_regr.tabpfn`
TabPFN Regression Learner

Description

Foundation model for tabular data. Uses **reticulate** to interface with the `tabpfn` Python package.

Installation

This learner relies on **reticulate** to handle Python dependencies. It is not necessary to install any Python package manually in advance or specify a Python environment via `reticulate::use_python()`, `reticulate::use_virtualenv()`, `reticulate::use_condaenv()`, or `reticulate::use_miniconda()`. By calling `$train()` or `$predict()`, the required Python packages (`tabpfn`, `torch`, etc.) will be installed automatically, if not already. Reticulate will then configure and initialize an ephemeral environment satisfying those requirements, unless an existing environment (e.g., `"r-reticulate"`) in reticulate's Order of Discovery contains all the necessary packages.

You may also manually install `tabpfn` into a Python environment following the official installation guide and specify the environment via `reticulate::use_*()` before calling `$train()` or `$predict()`. Note that the GPU version of PyTorch cannot be loaded by `reticulate::use_condaenv()` from a conda environment. To use a conda environment, please install the CPU version of PyTorch.

License acceptance

Since `tabpfn` version 8, the default model version (v3) requires a one-time acceptance of the Prior Labs license before the model weights can be downloaded. On the first call to `$train()` or `$predict()`, `tabpfn` opens a browser window to log in and accept the license, and caches the resulting token for subsequent calls. In non-interactive environments (e.g., on a server or in continuous integration), this browser flow fails. Instead, log in at `https://platform.priorlabs.ai`, accept the license on the licenses tab, copy your API key from `https://platform.priorlabs.ai/account`, and set it as the environment variable `TABPFN_TOKEN` before training.

Saving a Learner

In order to save a `LearnerRegrTabPFN` for later usage, it is necessary to call the `$marshal()` method on the `Learner` before writing it to disk, as the object will otherwise not be saved correctly. After loading a marshaled `LearnerRegrTabPFN` into R again, you then need to call `$unmarshal()` to transform it into a useable state.

Initial parameter values

- `n_jobs` is initialized to 1 to avoid threading conflicts with **future**.

Custom mlr3 parameters

- `output_type` corresponds to the same argument of the `.predict()` method of the `TabPFNRegressor` class, but only supports the options `"mean"`, `"median"` and `"mode"`. The point predictions are stored as `$response` of the prediction object.
- `categorical_features_indices` uses R indexing instead of zero-based Python indexing. It is only needed to mark numeric or logical features as categorical, because `factor`, `ordered`, and `character` features are always encoded as categorical by `tabpfn`. The level order of `ordered` features is not preserved, they are treated like unordered `factor` features.
- `device` must be a string. If set to `"auto"`, the behavior is the same as original. Otherwise, the string is passed as argument to `torch.device()` to create a device.
- `inference_precision` must be `"auto"`, `"autocast"`, or a `torch.dtype` string, e.g., `"torch.float32"`, `"torch.float64"`, etc. Non-float dtypes are not supported.
- `inference_config` is currently not supported.
- `n_jobs` is deprecated upstream in favor of `n_preprocessing_jobs` and is only kept for backward compatibility.
- `random_state` accepts either an integer or the special value `"None"` which corresponds to `None` in Python. Following the original Python implementation, the default `random_state` is `0`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("regr.tabpfn")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “quantiles”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **reticulate**

Parameters

Id	Type	Default	Levels
output_type	character	mean	mean, median, mode
n_estimators	integer	8	
auto_scale_n_estimators	logical	TRUE	TRUE, FALSE
categorical_features_indices	untyped	-	
softmax_temperature	numeric	0.9	
average_before_softmax	logical	FALSE	TRUE, FALSE
model_path	untyped	"auto"	
device	untyped	"auto"	
ignore_pretraining_limits	logical	FALSE	TRUE, FALSE
inference_precision	character	auto	auto, autocast, torch.float32, torch.float, torch.float64, torch.do

fit_mode	character	fit_preprocessors	low_memory, fit_preprocessors, fit_with_cache, batched
memory_saving_mode	untyped	"auto"	
keep_cache_on_device	logical	TRUE	TRUE, FALSE
random_state	integer	0	
n_jobs	integer	-	
n_preprocessing_jobs	integer	1	
differentiable_input	logical	FALSE	TRUE, FALSE
show_progress_bar	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrTabPFN`

Active bindings

`marshaled (logical(1))`
Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerRegrTabPFN$new()`
- `LearnerRegrTabPFN$marshal()`
- `LearnerRegrTabPFN$unmarshal()`
- `LearnerRegrTabPFN$clone()`

`LearnerRegrTabPFN$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerRegrTabPFN$new()
```

`LearnerRegrTabPFN$marshal()`: Marshal the learner's model.

Usage:

```
LearnerRegrTabPFN$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerRegrTabPFN$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerRegrTabPFN$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerRegrTabPFN$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerRegrTabPFN$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

b-zhou

References

Hollmann, Noah, Müller, Samuel, Purucker, Lennart, Krishnakumar, Arjun, Körfer, Max, Hoo, Bin S, Schirrmeister, Tibor R, Hutter, Frank (2025). "Accurate predictions on small data with a tabular foundation model." *Nature*. doi:10.1038/s41586024083286. https://www.nature.com/articles/s41586-024-08328-6.

Hollmann, Noah, Müller, Samuel, Eggensperger, Katharina, Hutter, Frank (2023). "TabPFN: A transformer that solves small tabular classification problems in a second." In *International Conference on Learning Representations 2023*.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

`mlr_learners_surv.aorsf`

Accelerated Oblique Random Survival Forest Learner

Description

Accelerated oblique random survival forest. Calls `aorsf::orsf()` from **aorsf**. Note that although the learner has the property `"missing"` and it can in principle deal with missing values, the behavior has to be configured using the parameter `na_action`.

Initial parameter values

- `n_thread`: This parameter is initialized to 1 (default is 0) to avoid conflicts with the mlr3 parallelization.
- `mtry`:
  - This hyperparameter can alternatively be set via the added hyperparameter `mtry_ratio` as `mtry = max(ceiling(mtry_ratio * n_features), 1)`. Note that `mtry` and `mtry_ratio` are mutually exclusive.

Prediction types

This learner returns three prediction types:

1. `distr`: a survival matrix in two dimensions, where observations are represented in rows and (unique event) time points in columns. Calculated using the internal `predict.ObliqueForest()` function.
2. `response`: the restricted mean survival time of each test observation, derived from the survival matrix prediction (`distr`).
3. `crank`: the expected mortality using `mlr3proba::surv_return()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.aorsf")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”, “response”
- Feature Types: “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **aorsf**

Parameters

Id	Type	Default	Levels	Range
n_tree	integer	500		$[1, \infty)$
n_split	integer	5		$[1, \infty)$
n_retry	integer	3		$[0, \infty)$
n_thread	integer	-		$[0, \infty)$
pred_aggregate	logical	TRUE	TRUE, FALSE	-
pred_simplify	logical	FALSE	TRUE, FALSE	-
oobag	logical	FALSE	TRUE, FALSE	-
mtry	integer	NULL		$[1, \infty)$
mtry_ratio	numeric	-		$[0, 1]$
sample_with_replacement	logical	TRUE	TRUE, FALSE	-
sample_fraction	numeric	0.632		$[0, 1]$

control_type	character	fast	fast, cph, net, custom	-
control_custom_fun	untyped	-		-
split_rule	character	logrank	logrank, cstat	-
control_fast_do_scale	logical	FALSE	TRUE, FALSE	-
control_fast_ties	character	efron	efron, breslow	-
control_cph_ties	character	efron	efron, breslow	-
control_cph_eps	numeric	1e-09		$[0, \infty)$
control_cph_iter_max	integer	20		$[1, \infty)$
control_net_alpha	numeric	0.5		$(-\infty, \infty)$
control_net_df_target	integer	NULL		$[1, \infty)$
leaf_min_events	integer	1		$[1, \infty)$
leaf_min_obs	integer	5		$[1, \infty)$
split_min_events	integer	5		$[1, \infty)$
split_min_obs	integer	10		$[1, \infty)$
split_min_stat	numeric	NULL		$[0, \infty)$
oobag_pred_type	character	risk	none, surv, risk, chf, mort	-
importance	character	anova	none, anova, negate, permute	-
importance_max_pvalue	numeric	0.01		$[1e-04, 0.9999]$
tree_seeds	integer	NULL		$[1, \infty)$
oobag_pred_horizon	numeric	NULL		$[0, \infty)$
oobag_eval_every	integer	NULL		$[1, \infty)$
oobag_fun	untyped	NULL		-
attach_data	logical	TRUE	TRUE, FALSE	-
verbose_progress	logical	FALSE	TRUE, FALSE	-
na_action	character	fail	fail, impute_meanmode	-

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvAorsf`

Methods

Public methods:

- `LearnerSurvAorsf$new()`
- `LearnerSurvAorsf$oob_error()`
- `LearnerSurvAorsf$importance()`
- `LearnerSurvAorsf$clone()`

`LearnerSurvAorsf$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvAorsf$new()
```

`LearnerSurvAorsf$oob_error()`: OOB concordance error extracted from the model slot `eval_oobag$stat_values`

Usage:

`LearnerSurvAorsf$oob_error()`

Returns: `numeric()`.

`LearnerSurvAorsf$importance()`: The importance scores are extracted from the model.

Usage:

`LearnerSurvAorsf$importance()`

Returns: Named `numeric()`.

`LearnerSurvAorsf$clone()`: The objects of this class are cloneable with this method.

Usage:

`LearnerSurvAorsf$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bcjaeger

References

Jaeger BC, Long DL, Long DM, Sims M, Szychowski JM, Min Y, Mcclure LA, Howard G, Simon N (2019). "Oblique random survival forests." *The Annals of Applied Statistics*, **13**(3). doi:10.1214/19aoas1261.

Jaeger BC, Welden S, Lenoir K, Speiser JL, Segar MW, Pandey A, Pajewski NM (2023). "Accelerated and interpretable oblique random survival forests." *Journal of Computational and Graphical Statistics*, 1–16. doi:10.1080/10618600.2023.2231048.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.aorsf")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_surv.bart

Survival Bayesian Additive Regression Trees Learner

Description

Fits a Bayesian Additive Regression Trees (BART) learner to right-censored survival data. Calls `BART::mc.surv.bart()` from **BART**.

Prediction types

This learner returns two prediction types:

1. `distr`: a 3d survival array with observations as 1st dimension, time points as 2nd and the posterior draws as 3rd dimension. Calculated using the internal `predict.survbart()` function.
2. `crank`: the expected mortality using `mlr3proba::surv_return()`. The parameter `which.curve` decides which posterior draw (3rd dimension) will be used for the calculation of the expected mortality. Note that the median posterior is by default used for the calculation of survival measures that require a `distr` prediction, see more info on PredictionSurv.

Initial parameter values

- `mc.cores` is initialized to 1 to avoid threading conflicts with **future**.

Custom mlr3 parameters

- `quiet` allows to suppress messages generated by the wrapped C++ code. Is initialized to `TRUE`.
- `importance` allows to choose the type of importance. Default is `count`, see documentation of method `$importance()` for more details.
- `which.curve` allows to choose which posterior draw will be used for the calculation of the `crank` prediction. If between (0,1) it is taken as the quantile of the curves otherwise if greater than 1 it is taken as the curve index, can also be 'mean'. By default the **median posterior** is used, i.e. `which.curve` is 0.5.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.bart")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3proba**, **BART**

Parameters

Id	Type	Default	Levels	Range
K	numeric	NULL		$[1, \infty)$
events	untyped	NULL		-
ztimes	untyped	NULL		-
zdelta	untyped	NULL		-
sparse	logical	FALSE	TRUE, FALSE	-
theta	numeric	0		$(-\infty, \infty)$
omega	numeric	1		$(-\infty, \infty)$
a	numeric	0.5		$[0.5, 1]$
b	numeric	1		$(-\infty, \infty)$
augment	logical	FALSE	TRUE, FALSE	-
rho	numeric	NULL		$(-\infty, \infty)$
usequants	logical	FALSE	TRUE, FALSE	-
rm.const	logical	TRUE	TRUE, FALSE	-
type	character	pbart	pbart, lbart	-
ntype	integer	-		$[1, 3]$
k	numeric	2		$[0, \infty)$
power	numeric	2		$[0, \infty)$
base	numeric	0.95		$[0, 1]$
offset	numeric	NULL		$(-\infty, \infty)$
ntree	integer	50		$[1, \infty)$
numcut	integer	100		$[1, \infty)$

ndpost	integer	1000		$[1, \infty)$
nskip	integer	250		$[0, \infty)$
keepevery	integer	10		$[1, \infty)$
printevery	integer	100		$[1, \infty)$
seed	integer	99		$(-\infty, \infty)$
mc.cores	integer	2		$[1, \infty)$
nice	integer	19		$[0, 19]$
openmp	logical	TRUE	TRUE, FALSE	-
quiet	logical	TRUE	TRUE, FALSE	-
importance	character	count	count, prob	-
which.curve	numeric	-		$[0, \infty)$

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvLearnerSurvBART`

Methods

Public methods:

- `LearnerSurvLearnerSurvBART$new()`
- `LearnerSurvLearnerSurvBART$importance()`
- `LearnerSurvLearnerSurvBART$clone()`

`LearnerSurvLearnerSurvBART$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvLearnerSurvBART$new()
```

`LearnerSurvLearnerSurvBART$importance()`: Two types of importance scores are supported based on the value of the parameter `importance`:

1. `prob`: The mean selection probability of each feature in the trees, extracted from the slot `varprob.mean`. If `sparse = FALSE` (default), this is a fixed constant. Recommended to use this option when `sparse = TRUE`.
2. `count`: The mean observed count of each feature in the trees (average number of times the feature was used in a tree decision rule across all posterior draws), extracted from the slot `varcount.mean`. This is the default importance scores.

In both cases, higher values signify more important variables.

Usage:

```
LearnerSurvLearnerSurvBART$importance()
```

Returns: Named `numeric()`.

`LearnerSurvLearnerSurvBART$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvLearnerSurvBART$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bblodfon

References

Sparapani, Rodney, Spanbauer, Charles, McCulloch, Robert (2021). "Nonparametric machine learning and efficient computation with bayesian additive regression trees: the BART R package." *Journal of Statistical Software*, **97**, 1–66.

Chipman, A H, George, I E, McCulloch, E R (2010). "BART: Bayesian additive regression trees." *The Annals of Applied Statistics*, **4**(1), 266–298.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
lrn("surv.bart")
```

---

mlr_learners_surv.blackboost

Gradient Boosting with Regression Trees Survival Learner

Description

Gradient boosting with regression trees for survival analysis. Calls `mboost::blackboost()` from **mboost**.

Details

`distr` prediction made by `mboost::survFit()`.

Prediction types

This learner returns two to three prediction types:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation. Calculated using `mboost::predict.blackboost()`. If the `family` parameter is not `"coxph"`, `-lp` is returned, since non-coxph families represent AFT-style distributions where lower `lp` values indicate higher risk.
2. `crank`: same as `lp`.
3. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using `mboost::survFit()`. This prediction type is present only when the `family` distribution parameter is equal to `"coxph"` (default). By default the Breslow estimator is used for computing the baseline hazard.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.blackboost")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr", "lp"
- Feature Types: "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **mboost**

Parameters

Id	Type	Default	Levels	Range
family	character	coxph	coxph, weibull, loglog, lognormal, gehan, cindex, custom	-
custom.family	untyped	-		-
nuirange	untyped	c(0, 100)		-
center	logical	TRUE	TRUE, FALSE	-
mstop	integer	100		$[0, \infty)$
nu	numeric	0.1		$[0, 1]$
risk	character	-	inbag, oobag, none	-
stopintern	logical	FALSE	TRUE, FALSE	-
trace	logical	FALSE	TRUE, FALSE	-
oobweights	untyped	-		-
teststat	character	quadratic	quadratic, maximum	-
splitstat	character	quadratic	quadratic, maximum	-
splittest	logical	FALSE	TRUE, FALSE	-
testtype	character	Bonferroni	Bonferroni, MonteCarlo, Univariate, Teststatistic	-
maxpts	integer	25000		$[1, \infty)$
abseps	numeric	0.001		$(-\infty, \infty)$
releps	numeric	0		$(-\infty, \infty)$
nmax	untyped	-		-

alpha	numeric	0.05		$[0, 1]$
mincriterion	numeric	0.95		$[0, 1]$
logmincriterion	numeric	-0.05129329		$(-\infty, 0]$
minsplit	integer	20		$[0, \infty)$
minbucket	integer	7		$[0, \infty)$
minprob	numeric	0.01		$[0, 1]$
stump	logical	FALSE	TRUE, FALSE	-
lookahead	logical	FALSE	TRUE, FALSE	-
MIA	logical	FALSE	TRUE, FALSE	-
nresample	integer	9999		$[1, \infty)$
tol	numeric	1.490116e-08		$[0, \infty)$
maxsurrogate	integer	0		$[0, \infty)$
mtry	integer	-		$[0, \infty)$
maxdepth	integer	-		$[0, \infty)$
multiway	logical	FALSE	TRUE, FALSE	-
splittry	integer	2		$[1, \infty)$
intersplit	logical	FALSE	TRUE, FALSE	-
majority	logical	FALSE	TRUE, FALSE	-
caseweights	logical	TRUE	TRUE, FALSE	-
sigma	numeric	0.1		$[0, 1]$
ipcw	untyped	1		-
na.action	untyped	stats::na.omit		-

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated via the `offset` argument in `mboost`'s training function. No offset is applied during prediction for this learner.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvBlackBoost`

Methods

Public methods:

- `LearnerSurvBlackBoost$new()`
- `LearnerSurvBlackBoost$clone()`

`LearnerSurvBlackBoost$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvBlackBoost$new()
```

`LearnerSurvBlackBoost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvBlackBoost$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

RaphaelS1

References

Bühlmann, Peter, Yu, Bin (2003). "Boosting with the L 2 loss: regression and classification." *Journal of the American Statistical Association*, **98**(462), 324–339.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.blackboost")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.blockforest`
BlockForest Survival Learner

Description

Random survival forests for blocks of clinical and omics covariate data. Calls `blockForest::blockfor()` from package **blockForest**.

In this learner, only the trained forest object (`$forest`) is retained. The optimized block-specific tuning parameters (`paramvalues`) and the biased OOB error estimate (`biased_oob_error_donotuse`) are discarded, as they are either not needed for downstream use or not reliable for performance estimation.

Prediction types

This learner returns two prediction types:

1. `distr`: a survival matrix in two dimensions, where observations are represented in rows and (unique event) time points in columns. Calculated using the internal `blockForest:::predict.blockForest()` function.
2. `crank`: the expected mortality using `mlr3proba::surv_return()`.

Initial parameter values

- `num.threads` is initialized to 1 to avoid conflicts with parallelization via **future**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.blockforest")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **blockForest**

Parameters

Id	Type	Default	Levels	Range
blocks	untyped	-		-
block.method	character	BlockForest	BlockForest, RandomBlock, BlockVarSel, VarProb, SplitWeights	-
num.trees	integer	2000		$[1, \infty)$

mtry	untyped	NULL		-
nsets	integer	300		$[1, \infty)$
num.trees.pre	integer	1500		$[1, \infty)$
splitrule	character	extratrees	logrank, extratrees, C, maxstat	-
always.select.block	integer	0		$[0, 1]$
importance	character	-	none, impurity, impurity_corrected, permutation	-
num.threads	integer	-		$[1, \infty)$
seed	integer	NULL		$(-\infty, \propto$
verbose	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvBlockForest`

Methods

Public methods:

- `LearnerSurvBlockForest$new()`
- `LearnerSurvBlockForest$importance()`
- `LearnerSurvBlockForest$clone()`

`LearnerSurvBlockForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvBlockForest$new()
```

`LearnerSurvBlockForest$importance()`: The importance scores are extracted from the model slot `variable.importance`.

Usage:

```
LearnerSurvBlockForest$importance()
```

Returns: Named `numeric()`.

`LearnerSurvBlockForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvBlockForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bblodfon

References

Hornung, R., Wright, N. M (2019). "Block Forests: Random forests for blocks of clinical and omics covariate data." *BMC Bioinformatics*, **20**(1), 1–17. doi:10.1186/s128590192942y. https://doi.org/10.1186/s12859-019-2942-y.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# check task's features
task$feature_names

# partition features to 2 blocks
blocks = list(bl1 = 1:3, bl2 = 4:6)

# define learner
learner = lrn("surv.blockforest", blocks = blocks,
              importance = "permutation", nsets = 10,
              num.trees = 50, num.trees.pre = 10, splitrule = "logrank")

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# feature importance
learner$importance()

# Make predictions for the test observations
pred = learner$predict(task, row_ids = ids$test)
pred

# Score the predictions
pred$score()
```

`mlr_learners_surv.cforest`
Survival Conditional Random Forest Learner

Description

A random forest based on conditional inference trees (`partykit::ctree()`). Calls `partykit::cforest()` from **partykit**.

Prediction types

This learner returns two prediction types:

1. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using the internal `partykit::predict.cforest()` function.
2. `crank`: the expected mortality using `mlr3proba::surv_return()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.cforest")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **partykit**, **sandwich**, **coin**, **survdistr**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
replace	logical	FALSE	TRUE, FALSE	-
fraction	numeric	0.632		$[0, 1]$
mtry	integer	-		$[0, \infty)$
mtryratio	numeric	-		$[0, 1]$
applyfun	untyped	-		-
cores	integer	NULL		$(-\infty, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
cluster	untyped	-		-
na.action	untyped	"stats::na.pass"		-
scores	untyped	-		-
teststat	character	quadratic	quadratic, maximum	-

splitstat	character	quadratic	quadratic, maximum	-
splittest	logical	FALSE	TRUE, FALSE	-
testtype	character	Univariate	Bonferroni, MonteCarlo, Univariate, Teststatistic	-
nmax	untyped	-		-
alpha	numeric	0.05		$[0, 1]$
mincriterion	numeric	0.95		$[0, 1]$
logmincriterion	numeric	-0.05129329		$(-\infty, \infty)$
minsplit	integer	20		$[1, \infty)$
minbucket	integer	7		$[1, \infty)$
minprob	numeric	0.01		$[0, 1]$
stump	logical	FALSE	TRUE, FALSE	-
lookahead	logical	FALSE	TRUE, FALSE	-
MIA	logical	FALSE	TRUE, FALSE	-
nresample	integer	9999		$[1, \infty)$
tol	numeric	1.490116e-08		$[0, \infty)$
maxsurrogate	integer	0		$[0, \infty)$
numsurrogate	logical	FALSE	TRUE, FALSE	-
maxdepth	integer	Inf		$[0, \infty)$
multiway	logical	FALSE	TRUE, FALSE	-
splittry	integer	2		$[0, \infty)$
intersplit	logical	FALSE	TRUE, FALSE	-
majority	logical	FALSE	TRUE, FALSE	-
caseweights	logical	TRUE	TRUE, FALSE	-
saveinfo	logical	FALSE	TRUE, FALSE	-
update	logical	FALSE	TRUE, FALSE	-
splitflavour	character	ctree	ctree, exhaustive	-
maxvar	integer	-		$[1, \infty)$
OOB	logical	FALSE	TRUE, FALSE	-
simplify	logical	TRUE	TRUE, FALSE	-
scale	logical	TRUE	TRUE, FALSE	-
maxpts	integer	25000		$(-\infty, \infty)$
abseps	numeric	0.001		$[0, \infty)$
releps	numeric	0		$[0, \infty)$

Initial parameter values

- `cores`: This parameter is initialized to 1 (default is `NULL`) to avoid threading conflicts with **future**.

Custom mlr3 parameters

- `mtry`:
  - This hyperparameter can alternatively be set via the added hyperparameter `mtryratio` as `mtry = max(ceiling(mtryratio * n_features), 1)`. Note that `mtry` and `mtryratio` are mutually exclusive.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvCForest`

Methods

Public methods:

- `LearnerSurvCForest$new()`
- `LearnerSurvCForest$clone()`

`LearnerSurvCForest$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvCForest$new()
```

`LearnerSurvCForest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvCForest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Hothorn T, Zeileis A (2015). "partykit: A Modular Toolkit for Recursive Partytioning in R." *Journal of Machine Learning Research*, **16**(118), 3905-3909. http://jmlr.org/papers/v16/hothorn15a.html.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. doi:10.1198/106186006x133933. https://doi.org/10.1198/106186006x133933.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.cforest", ntree = 50L)
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.coxboost`

Survival Cox Model with Likelihood Based Boosting Learner

Description

Fit a Survival Cox model with a likelihood based boosting algorithm. Calls `CoxBoost::CoxBoost()` from package **CoxBoost**.

Details

Use LearnerSurvCoxboost and LearnerSurvCVCoxboost for Cox boosting without and with internal cross-validation of boosting step number, respectively. Tuning using the internal optimizer in LearnerSurvCVCoxboost may be more efficient when tuning `stepno` only. However, for tuning multiple hyperparameters, **mlr3tuning** and LearnerSurvCoxboost will likely give better results.

Prediction types

This learner returns three prediction types, using the internal `predict.CoxBoost()` function:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation.
2. `crank`: same as `lp`.
3. `distr`: a 2d survival matrix, with observations as rows and time points as columns. The internal transformation uses the Breslow estimator to compute the baseline hazard and compose the survival distributions from the `lp` predictions.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.coxboost")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr", "lp"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **CoxBoost**

Parameters

Id	Type	Default	Levels	Range
unpen.index	untyped	-		-
standardize	logical	TRUE	TRUE, FALSE	-
stepno	integer	100		$[0, \infty)$
penalty	numeric	-		$(-\infty, \infty)$
criterion	character	pscore	pscore, score, hpscore, hscore	-
stepsize.factor	numeric	1		$(-\infty, \infty)$
sf.scheme	character	sigmoid	sigmoid, linear	-
pendistmat	untyped	-		-
connected.index	untyped	-		-
x.is.01	logical	FALSE	TRUE, FALSE	-
return.score	logical	TRUE	TRUE, FALSE	-
trace	logical	FALSE	TRUE, FALSE	-
at.step	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvCoxboost`

Methods

Public methods:

- `LearnerSurvCoxboost$new()`
- `LearnerSurvCoxboost$selected_features()`
- `LearnerSurvCoxboost$clone()`

`LearnerSurvCoxboost$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvCoxboost$new()
```

`LearnerSurvCoxboost$selected_features()`: Returns the set of selected features which have non-zero coefficients. Calls the internal `coef.CoxBoost()` function.

Usage:

```
LearnerSurvCoxboost$selected_features(at_step = NULL)
```

Arguments:

`at_step (integer(1))`
: Which boosting step to get the coefficients for. If no step is given (default), the final boosting step is used.

Returns: (`character()`) vector of feature names.

`LearnerSurvCoxboost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvCoxboost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Binder, Harald, Schumacher, Martin (2008). "Allowing for mandatory covariates in boosting estimation of sparse high-dimensional survival models." *BMC Bioinformatics*, **9**(1), 1-10. ISSN 14712105. doi:10.1186/14712105914.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.coxboost")
print(learner)
```

```
# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.ctree`

Survival Conditional Inference Tree Learner

Description

Survival Partition Tree where a significance test is used to determine the univariate splits. Calls `partykit::ctree()` from **partykit**.

Prediction types

This learner returns two prediction types:

1. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using the internal `partykit::predict.party()` function.
2. `crank`: the expected mortality using `mlr3proba::surv_return()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.ctree")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **partykit**, **coin**, **sandwich**, **survdistr**

Parameters

Id	Type	Default	Levels	Range
teststat	character	quadratic	quadratic, maximum	-
splitstat	character	quadratic	quadratic, maximum	-
splittest	logical	FALSE	TRUE, FALSE	-
testtype	character	Bonferroni	Bonferroni, MonteCarlo, Univariate, Teststatistic	-
nmax	untyped	-		-
alpha	numeric	0.05		$[0, 1]$
mincriterion	numeric	0.95		$[0, 1]$
logmincriterion	numeric	-		$(-\infty, \infty)$
minsplit	integer	20		$[1, \infty)$
minbucket	integer	7		$[1, \infty)$
minprob	numeric	0.01		$[0, \infty)$
stump	logical	FALSE	TRUE, FALSE	-
lookahead	logical	FALSE	TRUE, FALSE	-
MIA	logical	FALSE	TRUE, FALSE	-
nresample	integer	9999		$[1, \infty)$
tol	numeric	-		$[0, \infty)$
maxsurrogate	integer	0		$[0, \infty)$
numsurrogate	logical	FALSE	TRUE, FALSE	-
mtry	integer	Inf		$[0, \infty)$
maxdepth	integer	Inf		$[0, \infty)$
maxvar	integer	-		$[1, \infty)$
multiway	logical	FALSE	TRUE, FALSE	-
splittry	integer	2		$[0, \infty)$
intersplit	logical	FALSE	TRUE, FALSE	-
majority	logical	FALSE	TRUE, FALSE	-
caseweights	logical	FALSE	TRUE, FALSE	-
applyfun	untyped	-		-
cores	integer	NULL		$(-\infty, \infty)$
saveinfo	logical	TRUE	TRUE, FALSE	-
update	logical	FALSE	TRUE, FALSE	-
splitflavour	character	ctree	ctree, exhaustive	-
cluster	untyped	-		-
scores	untyped	-		-
doFit	logical	TRUE	TRUE, FALSE	-
maxpts	integer	25000		$(-\infty, \infty)$
abseps	numeric	0.001		$[0, \infty)$
releps	numeric	0		$[0, \infty)$

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvCTree`

Methods

Public methods:

- `LearnerSurvCTree$new()`
- `LearnerSurvCTree$clone()`

`LearnerSurvCTree$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvCTree$new()
```

`LearnerSurvCTree$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvCTree$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

adibender

References

Hothorn T, Zeileis A (2015). "partykit: A Modular Toolkit for Recursive Partytioning in R." *Journal of Machine Learning Research*, **16**(118), 3905-3909. http://jmlr.org/papers/v16/hothorn15a.html.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased Recursive Partitioning: A Conditional Inference Framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. doi:10.1198/106186006x133933. https://doi.org/10.1198/106186006x133933.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.ctree")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.cv_coxboost` *Survival Cox Model with Cross-Validation Likelihood Based Boosting Learner*

Description

Fits a survival Cox model using likelihood based boosting and internal cross-validation for the number of steps. Calls `CoxBoost::CoxBoost()` or `CoxBoost::cv.CoxBoost()` from package **CoxBoost**.

Details

Use LearnerSurvCoxboost and LearnerSurvCVCoxboost for Cox boosting without and with internal cross-validation of boosting step number, respectively. Tuning using the internal optimizer in LearnerSurvCVCoxboost may be more efficient when tuning `stepno` only. However, for tuning multiple hyperparameters, **mlr3tuning** and LearnerSurvCoxboost will likely give better results.

If `penalty == "optimCoxBoostPenalty"` then `CoxBoost::optimCoxBoostPenalty()` is used to determine the penalty value and optimal step to be used in `CoxBoost::CoxBoost()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.cv_coxboost")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr", "lp"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **CoxBoost**

Parameters

Id	Type	Default	Levels	Range
maxstepno	integer	100		$[0, \infty)$
K	integer	10		$[2, \infty)$
type	character	verweij	verweij, naive	-
folds	untyped	NULL		-
minstepno	integer	50		$[0, \infty)$
start.penalty	numeric	-		$(-\infty, \infty)$
iter.max	integer	10		$[1, \infty)$
upper.margin	numeric	0.05		$[0, 1]$
unpen.index	untyped	-		-
standardize	logical	TRUE	TRUE, FALSE	-
penalty	numeric	-		$(-\infty, \infty)$
criterion	character	pscore	pscore, score, hpscore, hscore	-
stepsize.factor	numeric	1		$(-\infty, \infty)$
sf.scheme	character	sigmoid	sigmoid, linear	-
pendistmat	untyped	-		-
connected.index	untyped	-		-
x.is.01	logical	FALSE	TRUE, FALSE	-
return.score	logical	TRUE	TRUE, FALSE	-
trace	logical	FALSE	TRUE, FALSE	-
at.step	untyped	-		-

Prediction types

This learner returns three prediction types, using the internal `predict.CoxBoost()` function:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation.
2. `crank`: same as `lp`.
3. `distr`: a 2d survival matrix, with observations as rows and time points as columns. The internal transformation uses the Breslow estimator to compute the baseline hazard and compose the survival distributions from the `lp` predictions.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvCVCoxboost`

Methods

Public methods:

- `LearnerSurvCVCoxboost$new()`
- `LearnerSurvCVCoxboost$selected_features()`
- `LearnerSurvCVCoxboost$clone()`

`LearnerSurvCVCoxboost$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvCVCoxboost$new()
```

`LearnerSurvCVCoxboost$selected_features()`: Returns the set of selected features which have non-zero coefficients. Calls the internal `coef.CoxBoost()` function.

Usage:

```
LearnerSurvCVCoxboost$selected_features(at_step = NULL)
```

Arguments:

`at_step (integer(1))`
Which boosting step to get the coefficients for. If no step is given (default), the final boosting step is used.

Returns: (`character()`) vector of feature names.

`LearnerSurvCVCoxboost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvCVCoxboost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Binder, Harald, Schumacher, Martin (2008). "Allowing for mandatory covariates in boosting estimation of sparse high-dimensional survival models." *BMC Bioinformatics*, **9**(1), 1-10. ISSN 14712105. doi:10.1186/14712105914.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.

- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
task = tsk("rats")
task$col_roles$feature = c("litter", "rx")
learner = lrn("surv.cv_coxboost", maxstepno = 20)
splits = partition(task)
learner$train(task, splits$train)
pred = learner$predict(task, splits$test)
```

`mlr_learners_surv.cv_glmnet`

Cross-Validated GLM with Elastic Net Regularization Survival Learner

---

Description

Generalized linear models with elastic net regularization. Calls `glmnet::cv.glmnet()` from package **glmnet**.

Initial parameter values

- `family` is set to `"cox"` and cannot be changed.
- `cox.ties` is initialized to `"breslow"` to keep the tie-handling behavior of earlier glmnet versions, and to silence the glmnet v5.0 warning about the upcoming default change to `"efron"`.

Prediction types

This learner returns three prediction types:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation. Calculated using `glmnet::predict.cv.glmnet()`.
2. `crank`: same as `lp`.
3. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using `glmnet:::survfit.cv.glmnet()`. Parameters `stype` and `ctype` relate to how `lp` predictions are transformed into survival predictions and are described in `survival::survfit.coxph()`. By default the Breslow estimator is used for computing the baseline hazard.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.cv_glmnet")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”, “lp”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **glmnet**, **survdistr**

Parameters

Id	Type	Default	Levels	Range
lambda	untyped	NULL		-
type.measure	character	deviance	deviance, C	-
nfolds	integer	10		$[3, \infty)$
foldid	untyped	NULL		-
alignment	character	lambda	lambda, fraction	-
grouped	logical	TRUE	TRUE, FALSE	-
keep	logical	FALSE	TRUE, FALSE	-
parallel	logical	FALSE	TRUE, FALSE	-
gamma	untyped	c(0, 0.25, 0.5, 0.75, 1)		-
relax	logical	FALSE	TRUE, FALSE	-
trace.it	integer	0		$[0, 1]$
alpha	numeric	1		$[0, 1]$
nlambda	integer	100		$[1, \infty)$
lambda.min.ratio	numeric	-		$[0, 1]$
standardize	logical	TRUE	TRUE, FALSE	-
intercept	logical	TRUE	TRUE, FALSE	-
exclude	untyped	NULL		-
penalty.factor	untyped	-		-
lower.limits	untyped	-Inf		-
upper.limits	untyped	Inf		-
cox.ties	character	breslow	breslow, efron	-
maxp	integer	-		$[1, \infty)$
path	logical	FALSE	TRUE, FALSE	-
fdev	numeric	1e-05		$[0, 1]$
devmax	numeric	0.999		$[0, 1]$
eps	numeric	1e-06		$[0, 1]$
big	numeric	9.9e+35		$(-\infty, \infty)$
mnlam	integer	5		$(-\infty, \infty)$
pmin	numeric	1e-09		$[0, 1]$
exmx	numeric	250		$(-\infty, \infty)$
prec	numeric	1e-10		$(-\infty, \infty)$
mxit	integer	100		$[1, \infty)$
epsnr	numeric	1e-06		$[0, 1]$
mxitnr	integer	25		$[1, \infty)$
thresh	numeric	1e-07		$[0, \infty)$
maxit	integer	100000		$[1, \infty)$
dfmax	integer	NULL		$[0, \infty)$
pmax	integer	NULL		$[0, \infty)$

s	numeric	lambda.1se		$[0, \infty)$
predict.gamma	numeric	gamma.1se		$[0, 1]$
exact	logical	FALSE	TRUE, FALSE	-
stype	integer	2		$[1, 2]$
ctype	integer	-		$[1, 2]$
use_pred_offset	logical	-	TRUE, FALSE	-
strata	untyped	NULL		-

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated during training via the `offset` argument in `glmnet::glmnet()`. During prediction, the offset column from the test set is used only if `use_pred_offset = TRUE` (default), passed via the `newoffset` argument in `glmnet::predict.glmnet()`. Otherwise, if the user sets `use_pred_offset = FALSE`, a zero offset is applied, effectively disabling the offset adjustment during prediction.

Stratification

Parameter `strata` can be set to the name of one task column used as the stratification variable. During training, the column is removed from the data, converted to an integer, and is attached to the survival response via `glmnet::stratifySurv()`. During prediction, the same task column is passed as `newstrata` for survival distribution predictions (`distr`) in the `survfit` function. Constant interpolation via `survdistr` is used to align the survival probabilities to a common set of time points across all strata. Note that if an observation in the test set has a stratum not present in the training set, the prediction will fail, as the baseline hazard cannot be estimated for unseen strata. The rest of the prediction types do not depend on the strata column.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvCVGlmnet`

Active bindings

`native_model (cv.glmnet)`
The fitted model.

Methods

Public methods:

- `LearnerSurvCVGlmnet$new()`
- `LearnerSurvCVGlmnet$selected_features()`
- `LearnerSurvCVGlmnet$clone()`

`LearnerSurvCVGlmnet$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvCVGlmnet$new()
```

`LearnerSurvCVGlmnet$selected_features()`: Returns the set of selected features as reported by `glmnet::predict.glmnet()` with `type` set to `"nonzero"`.

Usage:

```
LearnerSurvCVGlmnet$selected_features(lambda = NULL)
```

Arguments:

`lambda` (`numeric(1)`)
Custom `lambda`, defaults to the active lambda depending on parameter set.

Returns: (`character()`) of feature names.

`LearnerSurvCVGlmnet$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvCVGlmnet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

bblodfon

References

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **33**(1), 1–22. doi:10.18637/jss.v033.i01.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.cv_glmnet")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.cv_ncvsurv`

Cross-Validated MCP- or SCAD-penalized Survival Learner

Description

MCP- or SCAD-penalized regression survival model with k-fold cross validation for choosing the regularization parameter lambda. Calls `ncvreg::cv.ncvsurv()` from package **ncvreg**. Covariates are internally standardized by `ncvreg` prior to model fitting.

Prediction types

This learner returns three prediction types (by default the `lambda` corresponding to the minimum cross-validation error is used):

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation. Calculated using `ncvreg::predict.cv.ncvsurv()` with `type = "link"`.
2. `crank`: identical to `lp`.
3. `distr`: a survival probability matrix with dimensions `n_obs × n_times`, where rows correspond to observations and columns correspond to time points. Calculated using `ncvreg::predict.cv.ncvsurv()` with `type = "survival"`. By default the Kalbfleisch-Prentice estimator is used for the survival function and the time grid is defined by the unique time points in the training data.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.cv_ncvsurv")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”, “lp”
- Feature Types: “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **ncvreg**

Parameters

Id	Type	Default	Levels	Range
penalty	character	MCP	MCP, SCAD, lasso	-
gamma	numeric	-		$[0, \infty)$
alpha	numeric	1		$[0, 1]$
lambda.min	numeric	-		$[0, \infty)$
nlambda	integer	100		$[1, \infty)$
lambda	untyped	-		-
eps	numeric	1e-04		$(-\infty, \infty)$
max.iter	integer	10000		$[1, \infty)$
convex	logical	TRUE	TRUE, FALSE	-
dfmax	integer	-		$[0, \infty)$
penalty.factor	untyped	-		-
warn	logical	TRUE	TRUE, FALSE	-
returnX	logical	TRUE	TRUE, FALSE	-
cluster	untyped	-		-
nfolds	integer	10		$(-\infty, \infty)$
fold	untyped	-		-
se	character	quick	quick, bootstrap	-
returnY	logical	FALSE	TRUE, FALSE	-
trace	logical	FALSE	TRUE, FALSE	-
pred_lambda	numeric	-		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvNCVsurv`

Methods

Public methods:

- `LearnerSurvNCVsurv$new()`
- `LearnerSurvNCVsurv$selected_features()`
- `LearnerSurvNCVsurv$clone()`

`LearnerSurvNCVsurv$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvNCVsurv$new()
```

`LearnerSurvNCVsurv$selected_features()`: Returns the set of selected features which have non-zero coefficients. Calls the internal `coef.cv.ncvreg()` function.

Usage:

```
LearnerSurvNCVsurv$selected_features(lambda = NULL)
```

Arguments:

`lambda (numeric(1))`
Custom `lambda`, defaults to the lambda with the minimum CV error.

Returns: (`character()`) vector of feature names.

`LearnerSurvNCVsurv$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvNCVsurv$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bblodfon

References

Breheny P, Huang J (2011). "Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection." *Annals of Applied Statistics*, **5**(1), 232–253. ISSN 1932-6157. doi:10.1214/10AOAS388.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.cv_ncvsurv")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.flexreg`

Survival Flexible Parametric Learner

Description

Flexible parametric learner. Calls `flexsurv::flexsurvreg()` from **flexsurv**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.flexreg")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr", "lp", "response"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **flexsurv**, **survival**

Parameters

Id	Type	Default	Levels
formula	untyped	-	
anc	untyped	-	
bhazard	untyped	-	
rtrunc	untyped	-	
dist	character	-	gengamma, gengamma.orig, genf, genf.orig, weibull, weibullph, gamma, exp, exponenti
inits	untyped	-	
fixedpars	untyped	-	
cl	numeric	0.95	
hessian	logical	TRUE	TRUE, FALSE
hess.control	untyped	-	
maxiter	integer	30	
rel.tolerance	numeric	1e-09	
toler.chol	numeric	1e-10	
debug	integer	0	
outer.max	integer	10	
times	untyped	-	

Prediction types

This learner returns three prediction types:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation. Calculated using `predict.flexsurvreg(type = "lp", ...)`. The interpretation whether higher lp corresponds to higher risk or higher survival, depends on the distribution choice (see `dist` parameter documentation).
2. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using `predict.flexsurvreg(type = "survival", ...)`.
3. `crank`: same as `lp`.
4. `response`: mean survival time calculated using `predict.flexsurvreg(type = "response", ...)`.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvFlexReg`

Methods

Public methods:

- `LearnerSurvFlexReg$new()`
- `LearnerSurvFlexReg$clone()`

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

Usage:

```
LearnerSurvFlexReg$new()
```

`LearnerSurvFlexReg$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvFlexReg$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bblodfon

References

Jackson, H. C (2016). "flexsurv: A Platform for Parametric Survival Modeling in R." *Journal of Statistical Software*, **70**(8), 1–33. doi:10.18637/JSS.V070.I08. https://www.jstatsoft.org/index.php/jss/article/view/v070i08.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
library(survival)

# Define the task, split to train/test set
task = tsk("lung")
set.seed(42)
part = partition(task)

# Define the learner
learner = lrn("surv.flexreg",
  formula = survival::Surv(time, status) ~ age + ph.karno + sex,
  anc = list(shape = ~ sex), dist = "weibull")

# Train the learner on the training ids
learner$train(task, part$train)
print(learner$model)
```

```
# Make predictions for the test rows
predictions = learner$predict(task, part$test)
print(predictions)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.flexspline`
Survival Flexible Parametric Spline Learner

Description

Flexible survival regression using the Royston/Parmar spline model. Calls `flexsurv::flexsurvspline()` from **flexsurv**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.flexspline")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr", "lp", "response"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **flexsurv**, **survival**

Parameters

Id	Type	Default	Levels	Range
formula	untyped	-		-
bhazard	untyped	-		-
rtrunc	untyped	-		-
k	integer	0		$[0, \infty)$
knots	untyped	-		-
bknots	untyped	-		-
scale	character	hazard	hazard, odds, normal	-
timescale	character	log	log, identity	-
spline	character	rp	rp, splines2ns	-
inits	untyped	-		-
fixedpars	untyped	-		-
cl	numeric	0.95		$[0, 1]$

anc	untyped	-	-
maxiter	integer	30	$(-\infty, \infty)$
rel.tolerance	numeric	1e-09	$(-\infty, \infty)$
toler.chol	numeric	1e-10	$(-\infty, \infty)$
debug	integer	0	$[0, 1]$
outer.max	integer	10	$(-\infty, \infty)$
times	untyped	-	-

Prediction types

This learner returns three prediction types:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation. Calculated using `predict.flexsurvreg(type = "lp", ...)`. Higher linear predictor gives higher hazard (equivalently, lower survival) for the spline models.
2. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using `predict.flexsurvreg(type = "survival", ...)`.
3. `crank`: same as `lp`.
4. `response`: mean survival time calculated using `predict.flexsurvreg(type = "response", ...)`

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvFlexSpline`

Methods

Public methods:

- `LearnerSurvFlexSpline$new()`
- `LearnerSurvFlexSpline$clone()`

`LearnerSurvFlexSpline$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvFlexSpline$new()
```

`LearnerSurvFlexSpline$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvFlexSpline$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Royston, Patrick, Parmar, KB M (2002). "Flexible parametric proportional-hazards and proportional-odds models for censored survival data, with application to prognostic modelling and estimation of treatment effects." *Statistics in medicine*, **21**(15), 2175–2197.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
library(survival)

# Define the task, split to train/test set
task = tsk("lung")
set.seed(42)
part = partition(task)

# Define the learner
learner = lrn("surv.flexspline", k = 1,
  formula = survival::Surv(time, status) ~ age + ph.karno + sex,
  anc = list(gamma1 = ~ sex))

# Train the learner on the training ids
learner$train(task, part$train)
print(learner$model)

# Make predictions for the test rows
predictions = learner$predict(task, part$test)
print(predictions)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.gam.cox`
Cox Proportional Hazards Generalized Additive Learner

Description

Cox proportional hazards model fitted via generalized additive modeling. Internally calls `mgcv::gam()` from the **mgcv** package with `family = mgcv::cox.ph()`. The censoring indicator is passed via the `weights` argument.

Prediction types

Three types of prediction are returned for this learner:

1. `lp`: a vector of linear predictors (relative risk scores), one per observation. Calls `mgcv::predict.gam()` with `type = "link"`.
2. `crank`: same as `lp`.
3. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. By default, the Breslow estimator is used via `mlr3proba::breslow()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.gam.cox")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr", "lp"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **mgcv**

Parameters

Id	Type	Default	Levels	Range
formula	untyped	-		-
method	character	GCV.Cp	GCV.Cp, GACV.Cp, REML, P-REML, ML, P-ML	-
optimizer	untyped	c("outer", "newton")		-
scale	numeric	0		$(-\infty, \infty)$
select	logical	FALSE	TRUE, FALSE	-
knots	untyped	NULL		-
sp	untyped	NULL		-
min.sp	untyped	NULL		-
H	untyped	NULL		-

gamma	numeric	1		$[1, \infty)$
paraPen	untyped	NULL		-
G	untyped	NULL		-
in.out	untyped	NULL		-
drop.unused.levels	logical	TRUE	TRUE, FALSE	-
drop.intercept	logical	FALSE	TRUE, FALSE	-
nei	untyped	-		-
nthreads	integer	1		$[1, \infty)$
ncv.threads	integer	1		$[1, \infty)$
irls.reg	numeric	0		$[0, \infty)$
epsilon	numeric	1e-07		$[0, \infty)$
maxit	integer	200		$(-\infty, \infty)$
mgcv.tol	numeric	1e-07		$[0, \infty)$
mgcv.half	integer	15		$[0, \infty)$
trace	logical	FALSE	TRUE, FALSE	-
rank.tol	numeric	1.490116e-08		$[0, \infty)$
nlm	untyped	list()		-
optim	untyped	list()		-
newton	untyped	list()		-
idLinksBases	logical	TRUE	TRUE, FALSE	-
scalePenalty	logical	TRUE	TRUE, FALSE	-
efs.lspmax	integer	15		$[0, \infty)$
efs.tol	numeric	0.1		$[0, \infty)$
scale.est	character	fletcher	fletcher, pearson, deviance	-
edge.correct	logical	FALSE	TRUE, FALSE	-
block.size	integer	1000		$(-\infty, \infty)$
unconditional	logical	FALSE	TRUE, FALSE	-

Formula

A gam formula specific to the task at hand is required for the `formula` parameter (see example and `?mgcv::formula.gam`). Beware, if no formula is provided, a fallback formula is used that will make the gam behave like a glm (this behavior is required for the unit tests). Only features specified in the formula will be used, superseding columns with col_roles "feature" in the task.

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated during training via the `offset` argument in `mgcv::gam()`. No offset is applied during prediction for this learner.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvGamCox`

Methods

Public methods:

- `LearnerSurvGamCox$new()`
- `LearnerSurvGamCox$clone()`

`LearnerSurvGamCox$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvGamCox$new()
```

`LearnerSurvGamCox$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvGamCox$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

bblodfon

References

Wood, Simon (2012). "mgcv: Mixed GAM Computation Vehicle with GCV/AIC/REML smoothness estimation."

Wood, N. S, Pya, Natalya, Safken, Benjamin (2016). "Smoothing Parameter and Model Selection for General Smooth Models." *Journal of the American Statistical Association*, **111**(516), 1548–1563. doi:10.1080/01621459.2016.1180986.

Examples

```
# simple example
t = tsk("lung")
l = lrn("surv.gam.cox")
l$param_set$set_values(formula = time ~ s(age, k = 5) + ph.karno + sex)
l$train(t)
l$model
```

---

mlr_learners_surv.gamboost

Boosted Generalized Additive Survival Learner

Description

Fits a generalized additive survival model using a boosting algorithm. Calls `mboost::gamboost()` from **mboost**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.gamboost")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr", "lp"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **mboost**

Parameters

Id	Type	Default	Levels	Range
family	character	coxph	coxph, weibull, loglog, lognormal, gehan, cindex, custom	-
custom.family	untyped	-		-
nuirange	untyped	c(0, 100)		-
center	logical	TRUE	TRUE, FALSE	-
mstop	integer	100		$[0, \infty)$
nu	numeric	0.1		$[0, 1]$
risk	character	inbag	inbag, oobag, none	-
stopintern	untyped	FALSE		-
trace	logical	FALSE	TRUE, FALSE	-
oobweights	untyped	NULL		-
baselearner	character	bbs	bbs, bols, btree	-
dfbase	integer	4		$[0, \infty)$
sigma	numeric	0.1		$[0, 1]$
ipcw	untyped	1		-
na.action	untyped	stats::na.omit		-

Prediction types

This learner returns two to three prediction types:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation. Calculated using `mboost::predict.gamboost()`. If the `family` parameter is not `"coxph"`, `-lp` is returned, since non-coxph families represent AFT-style distributions where lower `lp` values indicate higher risk.
2. `crank`: same as `lp`.
3. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using `mboost::survFit()`. This prediction type is present only when the `family` distribution parameter is equal to `"coxph"` (default). By default the Breslow estimator is used for computing the baseline hazard.

Offset

If a Task contains a column with the offset role, it is automatically incorporated via the offset argument in mboost's training function. No offset is applied during prediction for this learner.

Super classes

mlr3::Learner -> mlr3proba::LearnerSurv -> LearnerSurvGAMBoost

Methods

Public methods:

- LearnerSurvGAMBoost$new()
- LearnerSurvGAMBoost$importance()
- LearnerSurvGAMBoost$selected_features()
- LearnerSurvGAMBoost$clone()

LearnerSurvGAMBoost$new(): Creates a new instance of this R6 class.

Usage:

```
LearnerSurvGAMBoost$new()
```

LearnerSurvGAMBoost$importance(): The importance scores are extracted with the function mboost::varimp() with the default arguments.

Usage:

```
LearnerSurvGAMBoost$importance()
```

Returns: Named numeric().

LearnerSurvGAMBoost$selected_features(): Selected features are extracted with the function mboost::variable.names.mboost(), with used.only = TRUE.

Usage:

```
LearnerSurvGAMBoost$selected_features()
```

Returns: character().

LearnerSurvGAMBoost$clone(): The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvGAMBoost$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Author(s)

RaphaelS1

References

Bühlmann, Peter, Yu, Bin (2003). "Boosting with the L 2 loss: regression and classification." *Journal of the American Statistical Association*, **98**(462), 324–339.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
lrn("surv.gamboost")
```

---

mlr_learners_surv.gbm *Survival Gradient Boosting Machine Learner*

Description

Gradient Boosting for Survival Analysis. Calls `gbm::gbm()` from **gbm**.

Prediction types

This learner returns two prediction types, using the internal `predict.gbm()` function:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation.
2. `crank`: same as `lp`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.gbm")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "lp"
- Feature Types: "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **gbm**

Parameters

Id	Type	Default	Levels	Range
distribution	character	coxph	coxph	-
n.trees	integer	100		$[1, \infty)$
cv.folds	integer	0		$[0, \infty)$
interaction.depth	integer	1		$[1, \infty)$
n.minobsinnode	integer	10		$[1, \infty)$
shrinkage	numeric	0.001		$[0, \infty)$
bag.fraction	numeric	0.5		$[0, 1]$
train.fraction	numeric	1		$[0, 1]$
keep.data	logical	FALSE	TRUE, FALSE	-
verbose	logical	FALSE	TRUE, FALSE	-
var.monotone	untyped	-		-
n.cores	integer	1		$(-\infty, \infty)$
single.tree	logical	FALSE	TRUE, FALSE	-

Initial parameter values

- `distribution`:
- Actual default: `"bernoulli"`
- Adjusted default: `"coxph"`
- Reason for change: This is the only distribution available for survival.
- `keep.data`:
  - Actual default: TRUE
  - Adjusted default: FALSE
  - Reason for change: `keep.data` = `FALSE` saves memory during model fitting.
- `n.cores`:
  - Actual default: NULL
  - Adjusted default: `1`
  - Reason for change: Suppressing the automatic internal parallelization if `cv.folds` > 0 and avoid threading conflicts with **future**.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvGBM`

Methods

Public methods:

- `LearnerSurvGBM$new()`
- `LearnerSurvGBM$importance()`

- `LearnerSurvGBM$clone()`

`LearnerSurvGBM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvGBM$new()
```

`LearnerSurvGBM$importance()`: The importance scores are extracted from the model slot `variable.importance`.

Usage:

```
LearnerSurvGBM$importance()
```

Returns: Named `numeric()`.

`LearnerSurvGBM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvGBM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Friedman, H J (2002). "Stochastic gradient boosting." *Computational statistics & data analysis*, **38**(4), 367–378.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.gbm")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_surv.glmboost

Boosted Generalized Linear Survival Learner

Description

Fits a generalized linear survival model using a boosting algorithm. Calls `mboost::glmboost()` from **mboost**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.glmboost")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr", "lp"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **mboost**

Parameters

Id	Type	Default	Levels	Range
family	character	coxph	coxph, weibull, loglog, lognormal, gehan, cindex, custom	-
custom.family	untyped	-		-
nuirange	untyped	c(0, 100)		-
center	logical	TRUE	TRUE, FALSE	-
mstop	integer	100		$[0, \infty)$
nu	numeric	0.1		$[0, 1]$
risk	character	inbag	inbag, oobag, none	-
oobweights	untyped	NULL		-
stopintern	logical	FALSE	TRUE, FALSE	-
trace	logical	FALSE	TRUE, FALSE	-
sigma	numeric	0.1		$[0, 1]$
ipcw	untyped	1		-
na.action	untyped	stats::na.omit		-
contrasts.arg	untyped	-		-

Prediction types

This learner returns two to three prediction types:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation. Calculated using `mboost::predict.glmboost()`. If the `family` parameter is not `"coxph"`, `-lp` is returned, since non-coxph families represent AFT-style distributions where lower `lp` values indicate higher risk.
2. `crank`: same as `lp`.
3. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using `mboost::survFit()`. This prediction type is present only when the `family` distribution parameter is equal to `"coxph"` (default). By default the Breslow estimator is used for computing the baseline hazard.

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated via the `offset` argument in `mboost`'s training function. No offset is applied during prediction for this learner.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvGLMBoost`

Methods

Public methods:

- `LearnerSurvGLMBoost$new()`

- LearnerSurvGLMBoost$importance()
- LearnerSurvGLMBoost$selected_features()
- LearnerSurvGLMBoost$clone()

`LearnerSurvGLMBoost$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvGLMBoost$new()
```

`LearnerSurvGLMBoost$importance()`: Importance scores are extracted with the function `mboost::varimp()` and represent a feature's individual contribution to the risk reduction per boosting step of the fitted model. The higher the risk reduction, the larger the feature importance.

Note: Importance is supported only for datasets with `numeric` features, as the presence of factors with multiple levels makes it difficult to get the original feature names.

Usage:

```
LearnerSurvGLMBoost$importance()
```

Returns: Named `numeric()`.

`LearnerSurvGLMBoost$selected_features()`: Selected features are extracted with the function `mboost::coef.glmboost()` which by default returns features with non-zero coefficients and for the final number of boosting iterations.

Note: Selected features can be retrieved only for datasets with `numeric` features, as the presence of factors with multiple levels makes it difficult to get the original feature names.

Usage:

```
LearnerSurvGLMBoost$selected_features()
```

Returns: `character()`.

`LearnerSurvGLMBoost$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvGLMBoost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Bühlmann, Peter, Yu, Bin (2003). "Boosting with the L 2 loss: regression and classification." *Journal of the American Statistical Association*, **98**(462), 324–339.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.glmboost")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.glmnet`

GLM with Elastic Net Regularization Survival Learner

Description

Generalized linear models with elastic net regularization. Calls `glmnet::glmnet()` from package **glmnet**.

Initial parameter values

- `family` is set to `"cox"` and cannot be changed.
- `cox.ties` is initialized to `"breslow"` to keep the tie-handling behavior of earlier glmnet versions, and to silence the glmnet v5.0 warning about the upcoming default change to `"efron"`.

Prediction types

This learner returns three prediction types:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation. Calculated using `glmnet::predict.glmnet()`.
2. `crank`: same as `lp`.
3. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using `glmnet:::survfit.coxnet()`. Parameters `stype` and `ctype` relate to how `lp` predictions are transformed into survival predictions and are described in `survival::survfit.coxph()`. By default the Breslow estimator is used for computing the baseline hazard.

Caution: This learner is different to learners calling `glmnet::cv.glmnet()` in that it does not use the internal optimization of parameter `lambda`. Instead, `lambda` needs to be tuned by the user (e.g., via **mlr3tuning**). When `lambda` is tuned, the `glmnet` will be trained for each tuning iteration. While fitting the whole path of `lambda`s would be more efficient, as is done by default in `glmnet::glmnet()`, tuning/selecting the parameter at prediction time (using parameter `s`) is currently not supported in **mlr3** (at least not in efficient manner). Tuning the `s` parameter is, therefore, currently discouraged.

When the data are i.i.d. and efficiency is key, we recommend using the respective auto-tuning counterpart in `mlr_learners_surv.cv_glmnet()`. However, in some situations this is not applicable, usually when data are imbalanced or not i.i.d. (longitudinal, time-series) and tuning requires custom resampling strategies (blocked design, stratification).

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated during training via the `offset` argument in `glmnet::glmnet()`. During prediction, the offset column from the test set is used only if `use_pred_offset` = `TRUE` (default), passed via the `newoffset` argument in `glmnet::predict.glmnet()`. Otherwise, if the user sets `use_pred_offset` = `FALSE`, a zero offset is applied, effectively disabling the offset adjustment during prediction.

Stratification

Parameter `strata` can be set to the name of one task column used as the stratification variable. During training, the column is removed from the data, converted to an integer, and is attached to the survival response via `glmnet::stratifySurv()`. During prediction, the same task column is passed as `newstrata` for survival distribution predictions (`distr`) in the `survfit` function. Constant interpolation via `survdistr` is used to align the survival probabilities to a common set of time points across all strata. Note that if an observation in the test set has a stratum not present in the training set, the prediction will fail, as the baseline hazard cannot be estimated for unseen strata. The rest of the prediction types do not depend on the strata column.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.glmnet")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”, “lp”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **glmnet**, **survdistr**

Parameters

Id	Type	Default	Levels	Range
alpha	numeric	1		$[0, 1]$
nlambda	integer	100		$[1, \infty)$
lambda.min.ratio	numeric	-		$[0, 1]$
lambda	untyped	NULL		-
standardize	logical	TRUE	TRUE, FALSE	-
intercept	logical	TRUE	TRUE, FALSE	-
exclude	untyped	NULL		-
penalty.factor	untyped	-		-
lower.limits	untyped	-Inf		-
upper.limits	untyped	Inf		-
relax	logical	FALSE	TRUE, FALSE	-
trace.it	integer	0		$[0, 1]$
cox.ties	character	breslow	breslow, efron	-
maxp	integer	-		$[1, \infty)$
path	logical	FALSE	TRUE, FALSE	-
fdev	numeric	1e-05		$[0, 1]$
devmax	numeric	0.999		$[0, 1]$
eps	numeric	1e-06		$[0, 1]$
big	numeric	9.9e+35		$(-\infty, \infty)$
mnlam	integer	5		$(-\infty, \infty)$
pmin	numeric	1e-09		$[0, 1]$
exmx	numeric	250		$(-\infty, \infty)$
prec	numeric	1e-10		$(-\infty, \infty)$
mxit	integer	100		$[1, \infty)$
epsnr	numeric	1e-06		$[0, 1]$
mxitnr	integer	25		$[1, \infty)$
thresh	numeric	1e-07		$[0, \infty)$
maxit	integer	100000		$[1, \infty)$
dfmax	integer	NULL		$[0, \infty)$
pmax	integer	NULL		$[0, \infty)$
exact	logical	FALSE	TRUE, FALSE	-
s	numeric	0.01		$[0, \infty)$

gamma	numeric	1		$[0, 1]$
stype	integer	2		$[1, 2]$
ctype	integer	-		$[1, 2]$
use_pred_offset	logical	-	TRUE, FALSE	-
strata	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvGlmnet`

Active bindings

`native_model` (`coxnet`)
The fitted model.

Methods

Public methods:

- `LearnerSurvGlmnet$new()`
- `LearnerSurvGlmnet$selected_features()`
- `LearnerSurvGlmnet$clone()`

`LearnerSurvGlmnet$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvGlmnet$new()
```

`LearnerSurvGlmnet$selected_features()`: Returns the set of selected features as reported by `glmnet::predict.glmnet()` with `type` set to `"nonzero"`.

Usage:

```
LearnerSurvGlmnet$selected_features(lambda = NULL)
```

Arguments:

`lambda` (`numeric(1)`)
Custom `lambda`, defaults to the active lambda depending on parameter set.

Returns: (`character()`) of feature names.

`LearnerSurvGlmnet$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvGlmnet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

bblodfon

References

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **33**(1), 1–22. doi:10.18637/jss.v033.i01.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.glmnet")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.mboost`

Boosted Generalized Additive Survival Learner

Description

Model-based boosting for survival analysis. Calls `mboost::mboost()` from **mboost**.

Details

`distr` prediction made by `mboost::survFit()`.

Prediction types

This learner returns two to three prediction types:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation. Calculated using `mboost::predict.mboost()`. If the `family` parameter is not `"coxph"`, `-lp` is returned, since non-coxph families represent AFT-style distributions where lower `lp` values indicate higher risk.
2. `crank`: same as `lp`.
3. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using `mboost::survFit()`. This prediction type is present only when the `family` distribution parameter is equal to `"coxph"` (default). By default the Breslow estimator is used for computing the baseline hazard.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.mboost")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”, “lp”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **mboost**

Parameters

Id	Type	Default	Levels	Range
family	character	coxph	coxph, weibull, loglog, lognormal, gehan, cindex, custom	-
custom.family	untyped	-		-
nuirange	untyped	c(0, 100)		-

center	logical	TRUE	TRUE, FALSE	-
mstop	integer	100		$[0, \infty)$
nu	numeric	0.1		$[0, 1]$
risk	character	inbag	inbag, oobag, none	-
stopintern	logical	FALSE	TRUE, FALSE	-
trace	logical	FALSE	TRUE, FALSE	-
oobweights	untyped	NULL		-
baselearner	character	bbs	bbs, bols, btree	-
sigma	numeric	0.1		$[0, 1]$
ipcw	untyped	1		-
na.action	untyped	stats::na.omit		-

Offset

If a `Task` contains a column with the `offset` role, it is automatically incorporated via the `offset` argument in `mboost`'s training function. No offset is applied during prediction for this learner.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvMBoost`

Methods

Public methods:

- `LearnerSurvMBoost$new()`
- `LearnerSurvMBoost$importance()`
- `LearnerSurvMBoost$selected_features()`
- `LearnerSurvMBoost$clone()`

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

Usage:

```
LearnerSurvMBoost$new()
```

`LearnerSurvMBoost$importance()`**:** The importance scores are extracted with the function `mboost::varimp()` with the default arguments.

Usage:

```
LearnerSurvMBoost$importance()
```

Returns: Named `numeric()`.

`LearnerSurvMBoost$selected_features()`**:** Selected features are extracted with the function `mboost::variable.names.mboost()`, with `used.only = TRUE`.

Usage:

```
LearnerSurvMBoost$selected_features()
```

Returns: `character()`.

`LearnerSurvMBoost$clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvMBoost$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Bühlmann, Peter, Yu, Bin (2003). "Boosting with the L 2 loss: regression and classification." *Journal of the American Statistical Association*, **98**(462), 324–339.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
lrn("surv.mboost")
```

`mlr_learners_surv.nelson`

Survival Nelson-Aalen Estimator Learner

Description

Non-parametric estimator of the cumulative hazard rate function. Calls `survival::survfit()` from **survival**.

Prediction types

This learner returns two prediction types:

1. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. The cumulative hazard $H(t)$ is calculated using the train data and the default parameters of the `survival::survfit()` function, i.e. `ctype = 1`, which uses the Nelson-Aalen formula. Then for each test observation the survival curve is $S(t) = \exp(-H(t))$.
2. `crank`: the expected mortality using `mlr3proba::surv_return()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.nelson")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **survival**

Parameters

Empty ParamSet

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvNelson`

Methods

Public methods:

- `LearnerSurvNelson$new()`
- `LearnerSurvNelson$clone()`

`LearnerSurvNelson$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvNelson$new()
```

`LearnerSurvNelson$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvNelson$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Nelson, Wayne (1969). "Hazard plotting for incomplete failure data." *Journal of Quality Technology*, **1**(1), 27–52.

Nelson, Wayne (1972). "Theory and applications of hazard plotting for censored failure data." *Technometrics*, **14**(4), 945–966.

Aalen, Odd (1978). "Nonparametric inference for a family of counting processes." *The Annals of Statistics*, 701–726.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.nelson")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.penalized`
Survival L1 and L2 Penalized Cox Learner

Description

Penalized (L1 and L2) Cox Proportional Hazards model. Calls `penalized::penalized()` from **penalized**.

Details

The `penalized` and `unpenalized` arguments in the learner are implemented slightly differently than in `penalized::penalized()`. Here, there is no parameter for `penalized` but instead it is assumed that every variable is penalized unless stated in the `unpenalized` parameter.

Prediction types

This learner returns two prediction types:

1. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using the internal `penalized::predict()` function. By default the Breslow estimator `penalized::breslow()` is used for computing the baseline hazard.
2. `crank`: the expected mortality using `mlr3proba::surv_return()`.

Initial parameter values

- `trace` is set to `"FALSE"` to disable printing output during model training.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.penalized")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **penalized**

Parameters

Id	Type	Default	Levels	Range
unpenalized	untyped	-		-
lambda1	untyped	0		-
lambda2	untyped	0		-
positive	logical	FALSE	TRUE, FALSE	-
fusedl	logical	FALSE	TRUE, FALSE	-
startbeta	numeric	-		$(-\infty, \infty)$
startgamma	numeric	-		$(-\infty, \infty)$
steps	integer	1		$[1, \infty)$
epsilon	numeric	1e-10		$[0, 1]$
maxiter	integer	-		$[1, \infty)$
standardize	logical	FALSE	TRUE, FALSE	-
trace	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvPenalized`

Methods

Public methods:

- `LearnerSurvPenalized$new()`
- `LearnerSurvPenalized$selected_features()`
- `LearnerSurvPenalized$clone()`

`LearnerSurvPenalized$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvPenalized$new()
```

`LearnerSurvPenalized$selected_features()`: Selected features are extracted with the method `coef()` of the S4 model object, see `penalized::penfit()`. By default it returns features with non-zero coefficients.

Note: Selected features can be retrieved only for datasets with `numeric` features, as the presence of factors with multiple levels makes it difficult to get the original feature names.

Usage:

```
LearnerSurvPenalized$selected_features()
```

Returns: `character()`.

`LearnerSurvPenalized$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvPenalized$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Goeman, J J (2010). "L1 penalized estimation in the Cox proportional hazards model." *Biometrical journal*, **52**(1), 70–84.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.penalized")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)


# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.priority_lasso`
Survival Priority Lasso Learner

Description

Patient outcome prediction based on multi-omics data taking practitioners' preferences into account. Calls `prioritylasso::prioritylasso()` from **prioritylasso**.

Prediction types

This learner returns three prediction types:

1. `lp`: a vector containing the linear predictors (relative risk scores), where each score corresponds to a specific test observation. Calculated using `prioritylasso::predict.prioritylasso()`.
2. `crank`: same as `lp`.
3. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. Calculated using `mlr3proba::breslow()` where the Breslow estimator is used for computing the baseline hazard.

Initial parameter values

- `family` is set to `"cox"` for the Cox survival objective and cannot be changed
- `type.measure` is set to `"deviance"` (cross-validation measure) and cannot be changed
- `cox.ties` is initialized to `"breslow"` to keep the tie-handling behavior of earlier glmnet versions, and to silence the glmnet v5.0 warning about the upcoming default change to `"efron"`

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.priority_lasso")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr", "lp"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3proba**, **prioritylasso**

Parameters

Id	Type	Default	Levels	Range
blocks	untyped	-		-
max.coef	untyped	NULL		-
block1.penalization	logical	TRUE	TRUE, FALSE	-
lambda.type	character	lambda.min	lambda.min, lambda.1se	-
standardize	logical	TRUE	TRUE, FALSE	-
nfolds	integer	10		$[3, \infty)$
foldid	untyped	NULL		-
cvoffset	logical	FALSE	TRUE, FALSE	-
cvoffsetnfolds	integer	10		$[1, \infty)$
return.x	logical	TRUE	TRUE, FALSE	-
lambda	untyped	NULL		-
grouped	logical	TRUE	TRUE, FALSE	-
trace.it	integer	0		$[0, 1]$
cox.ties	character	breslow	breslow, efron	-
include.allintercepts	logical	FALSE	TRUE, FALSE	-
use.blocks	untyped	"all"		-
adaptive.order	logical	FALSE	TRUE, FALSE	-

Scope and supported arguments

This learner intentionally exposes a focused subset of training and prediction arguments. It is designed to work well out of the box, without requiring extensive parameter tuning. Some arguments from `cv.glmnet()`, `glmnet()`, and `predict.prioritylasso()` are not included, because they are not consistently supported or forwarded through the full train/predict path (e.g. handling missing test data, or performing relaxed lasso fits).

Please open an issue if there is a need for supporting more learner parameters.

Custom mlr3 parameters

- `adaptive.order`: if `TRUE`, the priority order of blocks is estimated from the data following Herrmann et al. (2021), instead of using the user-supplied block order. For each block, a Ridge regression (`alpha = 0`) is fit using `cv.glmnet()` on that block alone. The importance of a block is measured by the mean absolute coefficient (MAC) score at the `lambda.min` value (the lambda giving minimum cross-validation error). A penalty factor of `1 / MAC` is then assigned to each block. Blocks are sorted by increasing penalty factor, i.e., blocks with larger MAC (stronger average signal) receive higher priority (come first). Also, the block-wise penalty factors are attached to the fitted model object as `learner$model$block.penalty.factors`.

This method is useful when no domain knowledge is available to specify block priority. In this step, data are **standardized by default** (`standardize = TRUE`), but this can be overridden by the learner's `standardize` parameter. `lambda.min` is always used to derive the block priority. Additional arguments such as `nfolds`, `type.measure`, `weights` and `cox.ties` (if provided) are forwarded to each

block-wise `cv.glmnet()` fit. The `max.coef` parameter, if supplied, it is re-ordered accordingly to align with the new block order.

This parameter is ignored when fewer than two blocks are provided. It defaults to `FALSE` for backward compatibility.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvPriorityLasso`

Methods

Public methods:

- `LearnerSurvPriorityLasso$new()`
- `LearnerSurvPriorityLasso$selected_features()`
- `LearnerSurvPriorityLasso$clone()`

`LearnerSurvPriorityLasso$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvPriorityLasso$new()
```

`LearnerSurvPriorityLasso$selected_features()`: Selected features, i.e. those where the coefficient is non-zero.

Usage:

```
LearnerSurvPriorityLasso$selected_features()
```

Returns: `character()`.

`LearnerSurvPriorityLasso$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvPriorityLasso$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

HarutyunyanLiana

bblodfon

References

Simon K, Vindi J, Roman H, Tobias H, Anne-Laure B (2018). "Priority-Lasso: a simple hierarchical approach to the prediction of clinical outcome using multi-omics data." *BMC Bioinformatics*, **19**. doi:10.1186/s1285901823446.

Herrmann, M., Probst, P., Hornung, R., Jurinovic, V., Boulesteix, L. A (2021). "Large-scale benchmark study of survival prediction methods using multi-omics data." *Briefings in Bioinformatics*, **22**(3), 1–15. doi:10.1093/BIB/BBAA167.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Check task's features
task$feature_names

# Partition features to 2 blocks
blocks = list(bl1 = 1:3, bl2 = 4:6)

# Define learner
learner = lrn("surv.priority_lasso", blocks = blocks, block1.penalization = FALSE,
              lambda.type = "lambda.1se", standardize = TRUE, nfolds = 5)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# Selected features
learner$selected_features()

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)
predictions

# Score the predictions
predictions$score()
```

`mlr_learners_surv.ranger`
Ranger Survival Learner

Description

Random survival forest. Calls `ranger::ranger()` from package **ranger**.

Prediction types

This learner returns two prediction types:

1. `distr`: a survival matrix in two dimensions, where observations are represented in rows and (unique event) time points in columns. Calculated using the internal `ranger::predict.ranger()` function.
2. `crank`: the expected mortality using `mlr3proba::surv_return()`.

Custom mlr3 parameters

- `mtry`: This hyperparameter can alternatively be set via our hyperparameter `mtry.ratio` as `mtry = max(ceiling(mtry.ratio * n_features), 1)`. Note that `mtry` and `mtry.ratio` are mutually exclusive.

Initial parameter values

- `num.threads` is initialized to 1 to avoid conflicts with parallelization via **future**.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.ranger")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **ranger**

Parameters

Id	Type	Default	Levels	Range
alpha	numeric	0.5		$(-\infty, \infty)$
always.split.variables	untyped	-		-
holdout	logical	FALSE	TRUE, FALSE	-
importance	character	-	none, impurity, impurity_corrected, permutation	-
keep.inbag	logical	FALSE	TRUE, FALSE	-
max.depth	integer	NULL		$[0, \infty)$
min.node.size	integer	5		$[1, \infty)$
minprop	numeric	0.1		$(-\infty, \infty)$
mtry	integer	-		$[1, \infty)$
mtry.ratio	numeric	-		$[0, 1]$
num.random.splits	integer	1		$[1, \infty)$
num.threads	integer	1		$[1, \infty)$
num.trees	integer	500		$[1, \infty)$
oob.error	logical	TRUE	TRUE, FALSE	-
regularization.factor	untyped	1		-
regularization.usedepth	logical	FALSE	TRUE, FALSE	-
replace	logical	TRUE	TRUE, FALSE	-
respect.unordered.factors	character	ignore	ignore, order, partition	-
sample.fraction	numeric	-		$[0, 1]$
save.memory	logical	FALSE	TRUE, FALSE	-
scale.permutation.importance	logical	FALSE	TRUE, FALSE	-
seed	integer	NULL		$(-\infty, \infty)$
split.select.weights	numeric	-		$[0, 1]$
splitrule	character	logrank	logrank, extratrees, C, maxstat	-
verbose	logical	TRUE	TRUE, FALSE	-
write.forest	logical	TRUE	TRUE, FALSE	-
min.bucket	integer	3		$(-\infty, \infty)$
time.interest	integer	NULL		$[1, \infty)$
node.stats	logical	FALSE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvRanger`

Methods

Public methods:

- `LearnerSurvRanger$new()`
- `LearnerSurvRanger$importance()`
- `LearnerSurvRanger$oob_error()`
- `LearnerSurvRanger$clone()`

`LearnerSurvRanger$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvRanger$new()
```

`LearnerSurvRanger$importance()`: The importance scores are extracted from the model slot `variable.importance`.

Usage:

```
LearnerSurvRanger$importance()
```

Returns: Named `numeric()`.

`LearnerSurvRanger$oob_error()`: The out-of-bag error is extracted from the model slot `prediction.error`.

Usage:

```
LearnerSurvRanger$oob_error()
```

Returns: `numeric(1)`.

`LearnerSurvRanger$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvRanger$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

be-marc

References

Wright, N. M, Ziegler, Andreas (2017). "ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R." *Journal of Statistical Software*, **77**(1), 1–17. doi:10.18637/jss.v077.i01.

Breiman, Leo (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565. doi:10.1023/A:1010933404324.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.ranger", importance = "permutation")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_surv.rfsrc

Survival Random Forest SRC Learner

Description

Random survival forest. Calls `randomForestSRC::rfsrc()` from **randomForestSRC**.

Prediction types

This learner returns two prediction types:

1. `distr`: a survival matrix in two dimensions, where observations are represented in rows and (unique event) time points in columns. Calculated using the internal `randomForestSRC::predict.rfsrc()` function.
2. `crank`: the expected mortality using `mlr3proba::surv_return()`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.rfsrc")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”
- Feature Types: “logical”, “integer”, “numeric”, “factor”
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **randomForestSRC**

Parameters

Id	Type	Default	Levels	Range
ntree	integer	500		$[1, \infty)$
mtry	integer	-		$[1, \infty)$
mtry.ratio	numeric	-		$[0, 1]$
nodesize	integer	15		$[1, \infty)$
nodedepth	integer	-		$[1, \infty)$
splitrule	character	logrank	logrank, bs.gradient	-
nsplit	integer	10		$[0, \infty)$
importance	character	FALSE	FALSE, TRUE, none, anti, permute, random	-
block.size	integer	10		$[1, \infty)$
bootstrap	character	by.root	by.root, by.node, none, by.user	-
samptype	character	swor	swor, swr	-
samp	untyped	-		-
membership	logical	FALSE	TRUE, FALSE	-
sampsize	untyped	-		-
sampsize.ratio	numeric	-		$[0, 1]$
na.action	character	na.omit	na.omit, na.impute	-
nimpute	integer	1		$[1, \infty)$
ntime	integer	150		$[0, \infty)$
proximity	character	FALSE	FALSE, TRUE, inbag, oob, all	-
distance	character	FALSE	FALSE, TRUE, inbag, oob, all	-
forest.wt	character	FALSE	FALSE, TRUE, inbag, oob, all	-
xvar.wt	untyped	-		-
split.wt	untyped	-		-
forest	logical	TRUE	TRUE, FALSE	-
var.used	character	FALSE	FALSE, all.trees	-
split.depth	character	FALSE	FALSE, all.trees, by.tree	-
seed	integer	-		$(-\infty, -1]$
do.trace	logical	FALSE	TRUE, FALSE	-
get.tree	untyped	-		-
outcome	character	train	train, test	-
ptn.count	integer	0		$[0, \infty)$
estimator	character	nelson	nelson, kaplan	-
cores	integer	1		$[1, \infty)$
use.uno	logical	TRUE	TRUE, FALSE	-
save.memory	logical	FALSE	TRUE, FALSE	-
perf.type	character	-	none	-
case.depth	logical	FALSE	TRUE, FALSE	-
marginal.xvar	untyped	NULL		-

Custom mlr3 parameters

- `estimator`: Hidden parameter that controls the type of estimator used to derive the survival function during prediction. The **default** value is `"chf"` which uses a bootstrapped Nelson-Aalen estimator for the cumulative hazard function $H(t)$, (Ishwaran, 2008) from which we calculate $S(t) = \exp(-H(t))$, whereas `"surv"` uses a bootstrapped Kaplan-Meier estimator to directly estimate $S(t)$.
- `mtry`: This hyperparameter can alternatively be set via the added hyperparameter `mtry.ratio` as `mtry = max(ceiling(mtry.ratio * n_features), 1)`. Note that `mtry` and `mtry.ratio` are mutually exclusive.
- `sampsize`: This hyperparameter can alternatively be set via the added hyperparameter `sampsize.ratio` as `sampsize = max(ceiling(sampsize.ratio * n_obs), 1)`. Note that `sampsize` and `sampsize.ratio` are mutually exclusive.
- `cores`: This value is set as the option `rf.cores` during training and is set to 1 by default.

Initial parameter values

- `ntime`: Number of time points to coerce the observed event times for use in the estimated survival function during prediction. We changed the default value of `150` to `0` in order to be in line with other random survival forest learners and use all the **unique event times from the train set**.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvRandomForestSRC`

Methods

Public methods:

- `LearnerSurvRandomForestSRC$new()`
- `LearnerSurvRandomForestSRC$importance()`
- `LearnerSurvRandomForestSRC$selected_features()`
- `LearnerSurvRandomForestSRC$oob_error()`
- `LearnerSurvRandomForestSRC$clone()`

`LearnerSurvRandomForestSRC$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvRandomForestSRC$new()
```

`LearnerSurvRandomForestSRC$importance()`: The importance scores are extracted from the model slot `importance`.

Usage:

`LearnerSurvRandomForestSRC$importance()`

Returns: Named `numeric()`.

`LearnerSurvRandomForestSRC$selected_features()`: Selected features are extracted from the model slot `var.used`.
Note: Due to a known issue in `randomForestSRC`, enabling `var.used = "all.trees"` causes prediction to fail. Therefore, this setting should be used exclusively for feature selection purposes and not when prediction is required.

Usage:

`LearnerSurvRandomForestSRC$selected_features()`

Returns: `character()`.

`LearnerSurvRandomForestSRC$oob_error()`: OOB error extracted from the model slot `err.rate`.

Usage:

`LearnerSurvRandomForestSRC$oob_error()`

Returns: `numeric()`.

`LearnerSurvRandomForestSRC$clone()`: The objects of this class are cloneable with this method.

Usage:

`LearnerSurvRandomForestSRC$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS (2008). "Random survival forests." *The Annals of Applied Statistics*, **2**(3). doi:10.1214/08aoas169. https://doi.org/10.1214/08-aoas169.

Breiman, Leo (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565. doi:10.1023/A:1010933404324.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.rfsrc", importance = "TRUE")
print(learner)

# Define a Task
task = tsk("grace")
# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_surv.survdnn

SurvDNN Survival Learner

Description

Deep neural network survival models from package **survdnn**, aimed at tabular (low to moderate-dimensional) covariate settings using torch-based multilayer perceptrons. The learner wraps `survdnn::survdnn()`.

Prediction types

This learner supports the following prediction types:

`lp` A numeric vector of linear predictors, one per observation. For `loss "cox"` / `"cox_l2"` this is a log-risk score (higher implies worse prognosis). For `"aft"`, `predict.survdnn()` returns the predicted log-time location $\mu(x)$ (higher implies better prognosis), therefore the learner internally negates it such that higher values imply higher risk (consistent with **mlr3** conventions). For `"coxtime"`, this is $g(t_0, x)$ evaluated at a reference time.

`crank` same as `lp`.

`distr` A survival matrix (rows = observations, columns = time points) based on `predict(type = "survival")`. By default, predictions are evaluated on the unique event times of the training data.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.survdnn")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “distr”, “lp”
- Feature Types: “integer”, “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **survdnn**, **torch**

Parameters

Id	Type	Default	Levels	Range
hidden	untyped	c(32L, 16L)		-
activation	character	relu	relu, leaky_relu, tanh, sigmoid, gelu, elu, softplus	-
lr	numeric	1e-04		$[1e-06, 1]$
epochs	integer	300		$[1, \infty)$
loss	character	cox	cox, cox_l2, aft, coxtime	-
optimizer	character	adam	adam, adamw, sgd, rmsprop, adagrad	-
optim_args	untyped	list()		-
verbose	logical	FALSE	TRUE, FALSE	-
dropout	numeric	0.3		$[0, 1]$
batch_norm	logical	TRUE	TRUE, FALSE	-
callbacks	untyped	NULL		-
.seed	integer	NULL		$(-\infty, \infty)$
.device	character	auto	auto, cpu, cuda	-
.threads	integer	NULL		$[1, \infty)$
na_action	character	omit	omit, fail	-

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvDNN`

Active bindings

`marshaled` (`logical(1)`) Whether the learner has been marshaled.

Methods

Public methods:

- `LearnerSurvDNN$new()`
- `LearnerSurvDNN$marshal()`

- `LearnerSurvDNN$unmarshal()`
- `LearnerSurvDNN$clone()`

`LearnerSurvDNN$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvDNN$new()
```

`LearnerSurvDNN$marshal()`: Marshal the learner's model.

Usage:

```
LearnerSurvDNN$marshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::marshal_model()`.

`LearnerSurvDNN$unmarshal()`: Unmarshal the learner's model.

Usage:

```
LearnerSurvDNN$unmarshal(...)
```

Arguments:

`...` (any)
Additional arguments passed to `mlr3::unmarshal_model()`.

`LearnerSurvDNN$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvDNN$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

ielbadisy

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.survdnn", epochs = 42L, loss = "cox")
print(learner)

# Define the task, split to train/test set
task = tsk("lung")
set.seed(42)
part = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = part$train)
print(learner$model)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = part$test)
print(predictions)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.svm` *Survival Support Vector Machine Learner*

Description

Survival support vector machine. Calls `survivalsvm::survivalsvm()` from **survivalsvm**.

Details

Four possible SVMs can be implemented, dependent on the `type` parameter. These correspond to predicting the survival time via regression (`regression`), predicting a continuous rank (`vanbelle1`, `vanbelle2`), or a hybrid of the two (`hybrid`). Whichever `type` is chosen determines how the `crank` predict type is calculated, but in any case all can be considered a valid continuous ranking.

`makediff3` is recommended when using `type = "hybrid"`.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.svm")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "response"
- Feature Types: "logical", "integer", "numeric", "character", "factor"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **survivalsvm**

Parameters

Id	Type	Default	Levels	Range
type	character	regression	regression, vanbelle1, vanbelle2, hybrid	-
diff.meth	character	-	makediff1, makediff2, makediff3	-
gamma	numeric	NULL		$(-\infty, \infty)$
mu	numeric	NULL		$(-\infty, \infty)$
opt.meth	character	quadprog	quadprog, ipop	-
kernel	character	lin_kernel	lin_kernel, add_kernel, rbf_kernel, poly_kernel	-
kernel.pars	untyped	-		-
sgf.sv	integer	5		$[0, \infty)$
sigf	integer	7		$[0, \infty)$
maxiter	integer	20		$[0, \infty)$
margin	numeric	0.05		$[0, \infty)$
bound	numeric	10		$[0, \infty)$
eig.tol	numeric	1e-06		$[0, \infty)$
conv.tol	numeric	1e-07		$[0, \infty)$
posd.tol	numeric	1e-08		$[0, \infty)$

Prediction types

This learner returns up to two prediction types:

1. `crank`: a vector containing the continuous ranking scores, where each score corresponds to a specific test observation.
2. `response`: the survival time of each test observation, equal to `-crank`. This prediction type if only available for `"type"` equal to `regression` or `hybrid`.

Custom mlr3 parameters

- `gamma`, `mu` have replaced `gamma.mu` so that it's easier to tune these separately. `mu` is only used when `type = "hybrid"`.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvSVM`

Methods

Public methods:

- `LearnerSurvSVM$new()`
- `LearnerSurvSVM$clone()`

`LearnerSurvSVM$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvSVM$new()
```

`LearnerSurvSVM$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvSVM$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

RaphaelS1

References

Van Belle, Vanya, Pelckmans, Kristiaan, Van Huffel, Sabine, Suykens, AK J (2011). "Improved performance on high-dimensional survival data by application of Survival-SVM." *Bioinformatics*, **27**(1), 87–94.

Van Belle, Vanya, Pelckmans, Kristiaan, Van Huffel, Sabine, Suykens, AK J (2011). "Support vector methods for survival analysis: a comparison between ranking and regression approaches." *Artificial intelligence in medicine*, **53**(2), 107–118.

Shivaswamy, K P, Chu, Wei, Jansche, Martin (2007). "A support vector approach to censored targets." In *Seventh IEEE international conference on data mining (ICDM 2007)*, 655–660. IEEE.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
set.seed(123)
# Define the Learner and set parameter values
learner = lrn("surv.svm", gamma = 0.1)
print(learner)

# Define a Task
task = tsk("rats")

# Create train and test set
```

```
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

# print the model
print(learner$model)

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

`mlr_learners_surv.xgboost.aft`

Extreme Gradient Boosting AFT Survival Learner

Description

eXtreme Gradient Boosting regression using an **Accelerated Failure Time** objective. Calls `xgboost::xgb.train()` from package **xgboost** with `objective` set to `survival:aft` and `eval_metric` to `aft-nloglik`.

Prediction types

This learner returns three prediction types:

1. `response`: the estimated survival time T for each test observation.
2. `lp`: a vector of linear predictors (relative risk scores), one per observation, estimated as $-log(T)$. Higher survival time denotes lower risk.
3. `crank`: same as `lp`.

Early stopping

Early stopping can be used to find the optimal number of boosting rounds. The `early_stopping_set` parameter controls which set is used to monitor the performance. By default, `early_stopping_set = "none"` which disables early stopping. Set `early_stopping_set = "test"` to monitor the performance of the model on the test set while training. The test set for early stopping can be set with the `"test"` row role in the mlr3::Task. Additionally, the range must be set in which the performance must increase with `early_stopping_rounds` and the maximum number of boosting rounds with `nrounds`. While resampling, the test set is automatically applied from the mlr3::Resampling. Not that using the test set for early stopping can potentially bias the performance scores.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.xgboost.aft")
```

Meta Information

- Task type: “surv”
- Predict Types: “crank”, “lp”, “response”
- Feature Types: “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **xgboost**

Parameters

Id	Type	Default	Levels	Range
aft_loss_distribution	character	normal	normal, logistic, extreme	-
aft_loss_distribution_scale	numeric	-		$(-\infty, \infty)$
alpha	numeric	0		$[0, \infty)$
base_score	numeric	0.5		$(-\infty, \infty)$
booster	character	gbtree	gbtree, gblinear, dart	-
callbacks	untyped	NULL		-
seed	integer	-		$(-\infty, \infty)$
colsample_bylevel	numeric	1		$[0, 1]$
colsample_bynode	numeric	1		$[0, 1]$
colsample_bytree	numeric	1		$[0, 1]$
disable_default_eval_metric	logical	FALSE	TRUE, FALSE	-
evals	untyped	NULL		-
early_stopping_rounds	integer	NULL		$[1, \infty)$
learning_rate	numeric	0.3		$[0, 1]$
feature_selector	character	cyclic	cyclic, shuffle, random, greedy, thrifty	-
gamma	numeric	0		$[0, \infty)$
grow_policy	character	depthwise	depthwise, lossguide	-
interaction_constraints	untyped	-		-
iterationrange	untyped	-		-
lambda	numeric	1		$[0, \infty)$
max_bin	integer	256		$[2, \infty)$
max_delta_step	numeric	0		$[0, \infty)$
max_depth	integer	6		$[0, \infty)$
max_leaves	integer	0		$[0, \infty)$
maximize	logical	NULL	TRUE, FALSE	-
min_child_weight	numeric	1		$[0, \infty)$
monotone_constraints	integer	0		$[-1, 1]$
normalize_type	character	tree	tree, forest	-
nrounds	integer	-		$[1, \infty)$
nthread	integer	-		$[1, \infty)$
num_parallel_tree	integer	1		$[1, \infty)$
one_drop	logical	FALSE	TRUE, FALSE	-
print_every_n	integer	1		$[1, \infty)$
rate_drop	numeric	0		$[0, 1]$
refresh_leaf	logical	TRUE	TRUE, FALSE	-
sampling_method	character	uniform	uniform, gradient_based	-
sample_type	character	uniform	uniform, weighted	-
save_name	untyped	-		-

save_period	integer	-		$[0, \infty)$
scale_pos_weight	numeric	1		$(-\infty, \infty)$
seed_per_iteration	logical	FALSE	TRUE, FALSE	-
skip_drop	numeric	0		$[0, 1]$
use_rmm	logical	FALSE	TRUE, FALSE	-
max_cached_hist_node	integer	NULL		$(-\infty, \infty)$
max_cat_to_onehot	integer	4		$(-\infty, \infty)$
max_cat_threshold	integer	64		$(-\infty, \infty)$
subsample	numeric	1		$[0, 1]$
top_k	integer	0		$[0, \infty)$
tree_method	character	auto	auto, exact, approx, hist, gpu_hist	-
updater	untyped	-		-
verbose	integer	-		$[0, 2]$
verbosity	integer	-		$[0, 2]$
xgb_model	untyped	-		-
device	untyped	-		-
missing	numeric	NA		$(-\infty, \infty)$
validate_features	logical	TRUE	TRUE, FALSE	-

Initial parameter values

- `nrounds` is initialized to 1000.
- `nthread` is initialized to 1 to avoid conflicts with parallelization via **future**.
- `verbose` and `verbosity` are both initialized to 0.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvXgboostAFT`

Active bindings

`internal_valid_scores` The last observation of the validation scores for all metrics. Extracted from `model$evaluation_log`

`best_valid_scores` The validation scores of the best boosting round, extracted from `model$evaluation_log`. Because XGBoost also predicts with the best `nrounds`, these are identical to `$internal_valid_scores` whenever early stopping is activated. If early stopping is not activated, no best round is tracked and this is an empty list.

`internal_tuned_values` Returns the early stopped iterations if `early_stopping_rounds` was set during training.

`validate` How to construct the internal validation data. This parameter can be either `NULL`, a ratio, `"test"`, or `"predefined"`.

`model` (any)
The fitted model. Only available after `$train()` has been called.

Methods

Public methods:

- `LearnerSurvXgboostAFT$new()`
- `LearnerSurvXgboostAFT$importance()`
- `LearnerSurvXgboostAFT$clone()`

`LearnerSurvXgboostAFT$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvXgboostAFT$new()
```

`LearnerSurvXgboostAFT$importance()`: The importance scores are calculated with `xgboost::xgb.importance()`.

Usage:

```
LearnerSurvXgboostAFT$importance()
```

Returns: Named `numeric()`.

`LearnerSurvXgboostAFT$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvXgboostAFT$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

To compute on GPUs, you first need to compile **xgboost** yourself and link against CUDA. See `https://xgboost.readthedocs.io/en/stable/build.html#building-with-gpu-support`.

Author(s)

bblodfon

References

Chen, Tianqi, Guestrin, Carlos (2016). "Xgboost: A scalable tree boosting system." In *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 785–794. ACM. doi:10.1145/2939672.2939785.

Avinash B, Hyunsu C, Toby H (2022). "Survival Regression with Accelerated Failure Time Model in XGBoost." *Journal of Computational and Graphical Statistics*. ISSN 15372715. doi:10.1080/10618600.2022.2067548.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`

- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.xgboost.aft")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

---

mlr_learners_surv.xgboost.cox

Extreme Gradient Boosting Cox Survival Learner

Description

eXtreme Gradient Boosting regression using a **Cox Proportional Hazards** objective. Calls `xgboost::xgb.train()` from package **xgboost** with `objective` set to `survival:cox` and `eval_metric` to `cox-nloglik`.

Prediction types

Three types of prediction are returned for this learner:

1. `lp`: a vector of linear predictors (relative risk scores), one per observation.
2. `crank`: same as `lp`.
3. `distr`: a survival matrix in two dimensions, where observations are represented in rows and time points in columns. By default, the Breslow estimator is used via `mlr3proba::breslow()`.

Initial parameter values

- `nrounds` is initialized to 1000.
- `nthread` is initialized to 1 to avoid conflicts with parallelization via **future**.
- `verbose` and `verbosity` are both initialized to 0.

Dictionary

This Learner can be instantiated via lrn():

```
lrn("surv.xgboost.cox")
```

Meta Information

- Task type: "surv"
- Predict Types: "crank", "distr", "lp"
- Feature Types: "integer", "numeric"
- Required Packages: **mlr3**, **mlr3proba**, **mlr3extralearners**, **xgboost**

Parameters

Id	Type	Default	Levels	Range
alpha	numeric	0		$[0, \infty)$
base_score	numeric	0.5		$(-\infty, \infty)$
booster	character	gbtree	gbtree, gblinear, dart	-
callbacks	untyped	list()		-
seed	integer	-		$(-\infty, \infty)$
colsample_bylevel	numeric	1		$[0, 1]$
colsample_bynode	numeric	1		$[0, 1]$
colsample_bytree	numeric	1		$[0, 1]$
disable_default_eval_metric	logical	FALSE	TRUE, FALSE	-
early_stopping_rounds	integer	NULL		$[1, \infty)$
evals	untyped	NULL		-
learning_rate	numeric	0.3		$[0, 1]$
feature_selector	character	cyclic	cyclic, shuffle, random, greedy, thrifty	-
gamma	numeric	0		$[0, \infty)$
grow_policy	character	depthwise	depthwise, lossguide	-
interaction_constraints	untyped	-		-
iterationrange	untyped	-		-
lambda	numeric	1		$[0, \infty)$
max_bin	integer	256		$[2, \infty)$
max_delta_step	numeric	0		$[0, \infty)$
max_depth	integer	6		$[0, \infty)$
max_leaves	integer	0		$[0, \infty)$
maximize	logical	NULL	TRUE, FALSE	-
min_child_weight	numeric	1		$[0, \infty)$
monotone_constraints	integer	0		$[-1, 1]$

normalize_type	character	tree	tree, forest	-
nrounds	integer	-		$[1, \infty)$
nthread	integer	-		$[1, \infty)$
num_parallel_tree	integer	1		$[1, \infty)$
one_drop	logical	FALSE	TRUE, FALSE	-
print_every_n	integer	1		$[1, \infty)$
rate_drop	numeric	0		$[0, 1]$
refresh_leaf	logical	TRUE	TRUE, FALSE	-
sampling_method	character	uniform	uniform, gradient_based	-
sample_type	character	uniform	uniform, weighted	-
save_name	untyped	-		-
save_period	integer	-		$[0, \infty)$
scale_pos_weight	numeric	1		$(-\infty, \infty)$
seed_per_iteration	logical	FALSE	TRUE, FALSE	-
skip_drop	numeric	0		$[0, 1]$
use_rmm	logical	FALSE	TRUE, FALSE	-
max_cached_hist_node	integer	NULL		$(-\infty, \infty)$
max_cat_to_onehot	integer	4		$(-\infty, \infty)$
max_cat_threshold	integer	64		$(-\infty, \infty)$
subsample	numeric	1		$[0, 1]$
top_k	integer	0		$[0, \infty)$
tree_method	character	auto	auto, exact, approx, hist, gpu_hist	-
updater	untyped	-		-
verbose	integer	-		$[0, 2]$
verbosity	integer	-		$[0, 2]$
xgb_model	untyped	-		-
device	untyped	-		-
missing	numeric	NA		$(-\infty, \infty)$
validate_features	logical	TRUE	TRUE, FALSE	-

Early stopping

Early stopping can be used to find the optimal number of boosting rounds. The `early_stopping_set` parameter controls which set is used to monitor the performance. By default, `early_stopping_set = "none"` which disables early stopping. Set `early_stopping_set = "test"` to monitor the performance of the model on the test set while training. The test set for early stopping can be set with the `"test"` row role in the mlr3::Task. Additionally, the range must be set in which the performance must increase with `early_stopping_rounds` and the maximum number of boosting rounds with `nrounds`. While resampling, the test set is automatically applied from the mlr3::Resampling. Not that using the test set for early stopping can potentially bias the performance scores.

Super classes

`mlr3::Learner` -> `mlr3proba::LearnerSurv` -> `LearnerSurvXgboostCox`

Active bindings

`internal_valid_scores` (named `list()` or `NULL`) Validation metrics extracted from the xgboost model's `evaluation_log`. If early stopping is enabled, the scores correspond to the `best_iteration` selected by early stopping. Otherwise, the scores are taken from the final boosting round (`nrounds`).

`best_valid_scores` (named `list()` or `NULL`) Validation metrics of the best boosting round, extracted from the xgboost model's `evaluation_log`. Because XGBoost also predicts with the `best_iteration`, these are identical to `$internal_valid_scores` whenever early stopping is enabled. If early stopping is not enabled, no best round is tracked and this is an empty list.

`internal_tuned_values` (named `list()` or `NULL`) If early stopping is activated, this returns a list with the early stopped iterations (`nrounds`), which is extracted from the best iteration of the model. Otherwise `NULL`.

`validate` How to construct the internal validation data. This parameter can be either `NULL`, a ratio, `"test"`, or `"predefined"`.

`model` (any)
The fitted model. Only available after `$train()` has been called.

Methods

Public methods:

- `LearnerSurvXgboostCox$new()`
- `LearnerSurvXgboostCox$importance()`
- `LearnerSurvXgboostCox$clone()`

`LearnerSurvXgboostCox$new()`: Creates a new instance of this R6 class.

Usage:

```
LearnerSurvXgboostCox$new()
```

`LearnerSurvXgboostCox$importance()`: The importance scores are calculated with `xgboost::xgb.importance()`.

Usage:

```
LearnerSurvXgboostCox$importance()
```

Returns: Named `numeric()`.

`LearnerSurvXgboostCox$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvXgboostCox$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Note

To compute on GPUs, you first need to compile **xgboost** yourself and link against CUDA. See https://xgboost.readthedocs.io/en/stable/build.html#building-with-gpu-support.

Author(s)

bblodfon

References

Chen, Tianqi, Guestrin, Carlos (2016). "Xgboost: A scalable tree boosting system." In *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 785–794. ACM. doi:10.1145/2939672.2939785.

See Also

- Dictionary of Learners: mlr3::mlr_learners.
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- **mlr3learners** for a selection of recommended learners.
- **mlr3cluster** for unsupervised clustering learners.
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Examples

```
# Define the Learner
learner = lrn("surv.xgboost.cox")
print(learner)

# Define a Task
task = tsk("grace")

# Create train and test set
ids = partition(task)

# Train the learner on the training ids
learner$train(task, row_ids = ids$train)

print(learner$model)
print(learner$importance())

# Make predictions for the test rows
predictions = learner$predict(task, row_ids = ids$test)

# Score the predictions
predictions$score()
```

Index

assert_python_packages, 7

caret::plsda(), *181*
class::lvq1(), *142*
class::lvqinit(), *142*
class::lvqtest(), *142*
classif.gam, *23*

data.table::subset, *8*
deepnet::sae.dnn.train(), *211*
Dictionary, *11*, *13*, *16*, *19*, *22*, *25*, *28*, *31*, *34*, *37*, *43*, *47*, *50*, *52*, *55*, *58*, *61*, *63*, *66*, *68*, *71*, *74*, *76*, *81*, *83*, *86*, *88*, *91*, *95*, *98*, *101*, *104*, *107*, *111*, *114*, *117*, *119*, *122*, *125*, *127*, *133*, *136*, *139*, *142*, *144*, *146*, *148*, *151*, *154*, *157*, *160*, *163*, *166*, *168*, *171*, *174*, *177*, *180*, *182*, *184*, *187*, *191*, *194*, *196*, *199*, *202*, *205*, *208*, *210*, *213*, *215*, *217*, *220*, *223*, *227*, *229*, *232*, *236*, *239*, *242*, *245*, *249*, *252*, *255*, *258*, *261*, *264*, *266*, *269*, *271*, *274*, *282*, *285*, *287*, *290*, *293*, *297*, *301*, *304*, *307*, *309*, *312*, *315*, *318*, *321*, *323*, *325*, *327*, *329*, *334*, *337*, *340*, *342*, *345*, *347*, *349*, *352*, *355*, *358*, *361*, *364*, *367*, *370*, *373*, *375*, *378*, *383*, *386*, *389*, *392*, *395*, *397*, *400*, *403*, *405*, *408*, *410*, *413*, *415*, *417*, *420*, *423*, *426*, *429*, *431*, *434*, *438*, *441*, *443*, *445*, *449*, *452*, *455*, *459*, *463*, *466*, *470*, *473*, *476*, *479*, *482*, *485*, *488*, *492*, *495*, *498*, *501*, *507*, *509*, *513*, *517*, *520*, *522*, *525*, *529*, *532*, *536*, *539*, *542*, *546*, *551*

GPfit::GP_fit(), *348*

h2o::h2o.deeplearning(), *92*, *353*
h2o::h2o.gbm(), *95*, *356*
h2o::h2o.glm(), *98*, *359*, *361*
h2o::h2o.init(), *99*, *102*, *356*, *359*, *362*
h2o::h2o.randomForest(), *102*, *362*

install_learners, 8

Learner, *9*, *12*, *14*, *17*, *20*, *24*, *26*, *30*, *32*, *35*, *38*, *39*, *44*, *48*, *51*, *53*, *56*, *59*, *62*, *65*, *69*, *72*, *75*, *77*, *79*, *82*, *84*, *87*, *89*, *92*, *96*, *99*, *102*, *105*, *108*, *112*, *115*, *118*, *121*, *123*, *126*, *128*, *134*, *137*, *140*, *143*, *145*, *147*, *149*, *152*, *155*, *158*, *161*, *164*, *167*, *169*, *173*, *175*, *178*, *181*, *183*, *185*, *189*, *192*, *195*, *197*, *200*, *203*, *206*, *209*, *211*, *214*, *216*, *218*, *222*, *225*, *228*, *230*, *233*, *237*, *240*, *243*, *246*, *250*, *253*, *257*, *259*, *262*, *265*, *267*, *270*, *273*, *275*, *277*, *279*, *281*, *283*, *286*, *289*, *291*, *294*, *298*, *302*, *305*, *308*, *310*, *313*, *316*, *319*, *324*, *326*, *328*, *331*, *333*, *335*, *338*, *341*, *343*, *346*, *348*, *350*, *353*, *356*, *359*, *362*, *365*, *368*, *371*, *374*, *376*, *379*, *384*, *387*, *390*, *393*, *396*, *398*, *401*, *403*, *406*, *409*, *411*, *414*, *415*, *418*, *421*, *424*, *427*, *430*, *433*, *435*, *439*, *442*, *444*, *447*, *450*, *453*, *457*, *461*, *464*, *468*, *471*, *474*, *477*, *481*, *483*, *486*, *489*, *494*, *496*, *499*, *502*, *505*, *507*, *510*, *515*, *518*, *521*, *523*, *526*, *530*, *533*, *538*, *540*, *543*, *548*
LearnerClassifAbess *(*mlr_learners_classif.abess*)*, 9
LearnerClassifAdabag *(*mlr_learners_classif.adabag*)*, 14
LearnerClassifAdaBoosting *(*mlr_learners_classif.ada*)*, 11
LearnerClassifAdaBoostM1

(mlr_learners_classif.AdaBoostM1), 17
LearnerClassifBam (mlr_learners_classif.bam), 23
LearnerClassifBart (mlr_learners_classif.bart), 26
LearnerClassifBayesNet (mlr_learners_classif.bayes_net), 29
LearnerClassifBlockForest (mlr_learners_classif.blockforest), 32
LearnerClassifBst (mlr_learners_classif.bst), 35
LearnerClassifC50 (mlr_learners_classif.C50), 38
LearnerClassifCatboost (mlr_learners_classif.catboost), 39
LearnerClassifCForest (mlr_learners_classif.cforest), 44
LearnerClassifCTree (mlr_learners_classif.ctree), 47
LearnerClassifDbnDNN (mlr_learners_classif.dbnDNN), 53
LearnerClassifDecisionStump (mlr_learners_classif.decision_stump), 55
LearnerClassifDecisionTable (mlr_learners_classif.decision_table), 58
LearnerClassifDiagLda (mlr_learners_classif.dlda), 62
LearnerClassifEarth (mlr_learners_classif.earth), 64
LearnerClassifEvtree (mlr_learners_classif.evtree), 67
LearnerClassifExhaustiveSearch (mlr_learners_classif.exhaustive_search), 69
LearnerClassifFastai (mlr_learners_classif.fastai), 71
LearnerClassifFNN (mlr_learners_classif.fnn), 75
LearnerClassifGam (mlr_learners_classif.gam), 77
LearnerClassifGAMBoost (mlr_learners_classif.gamboost), 79
LearnerClassifGausspr (mlr_learners_classif.gausspr), 82
LearnerClassifGBM (mlr_learners_classif.gbm), 84
LearnerClassifGLMBoost (mlr_learners_classif.glmboost), 87
LearnerClassifGlmer (mlr_learners_classif.glmer), 89
LearnerClassifH2ODeeplearning (mlr_learners_classif.h2o.deeplearning), 92
LearnerClassifH2OGBM (mlr_learners_classif.h2o.gbm), 95
LearnerClassifH2OGLM (mlr_learners_classif.h2o.glm), 98
LearnerClassifH2ORandomForest (mlr_learners_classif.h2o.randomForest), 102
LearnerClassifIBk (mlr_learners_classif.IBk), 105
LearnerClassifImbalancedRandomForestSRC (mlr_learners_classif.imbalanced_rfsrc), 108
LearnerClassifJ48 (mlr_learners_classif.J48), 112
LearnerClassifJRip (mlr_learners_classif.JRip), 115
LearnerClassifKnn (mlr_learners_classif.knn), 118
LearnerClassifKStar (mlr_learners_classif.kstar), 120
LearnerClassifKSVM (mlr_learners_classif.ksvm), 123

LearnerClassifLiblineaR
(mlr_learners_classif.liblinear), 125
LearnerClassifLightGBM
(mlr_learners_classif.lightgbm), 128
LearnerClassifLMT
(mlr_learners_classif.LMT), 134
LearnerClassifLogistic
(mlr_learners_classif.logistic), 137
LearnerClassifLSSVM
(mlr_learners_classif.lssvm), 140
LearnerClassifLvq1
(mlr_learners_classif.lvq1), 142
LearnerClassifMda
(mlr_learners_classif.mda), 145
LearnerClassifMdeb
(mlr_learners_classif.mdeb), 147
LearnerClassifMLPRsnns
(mlr_learners_classif.mlp_rsnns), 149
LearnerClassifMob
(mlr_learners_classif.mob), 152
LearnerClassifMultilayerPerceptron
(mlr_learners_classif.multilayer_perceptron), 155
LearnerClassifNaiveBayesMultinomial
(mlr_learners_classif.naive_bayes_multinomial), 158
LearnerClassifNaiveBayesWeka
(mlr_learners_classif.naive_bayes_weka), 161
LearnerClassifNCVreg
(mlr_learners_classif.cv_ncvreg), 50
LearnerClassifNeuralnet
(mlr_learners_classif.neuralnet), 164
LearnerClassifNNTrain
(mlr_learners_classif.nnTrain), 167
LearnerClassifObliqueRandomForest
(mlr_learners_classif.aorsf), 20
LearnerClassifOneR
(mlr_learners_classif.OneR), 169
LearnerClassifOPLS
(mlr_learners_classif.opls), 172
LearnerClassifPART
(mlr_learners_classif.PART), 175
LearnerClassifPenalized
(mlr_learners_classif.penalized), 178
LearnerClassifPlsdaCaret
(mlr_learners_classif.plsdaCaret), 181
LearnerClassifPolyFit
(mlr_learners_classif.polyFit), 183
LearnerClassifPriorityLasso
(mlr_learners_classif.priority_lasso), 184
LearnerClassifRandomForest
(mlr_learners_classif.randomForest), 194
LearnerClassifRandomForestSRC
(mlr_learners_classif.rfsrc), 205
LearnerClassifRandomForestWeka
(mlr_learners_classif.random_forest_weka), 188
LearnerClassifRandomTree
(mlr_learners_classif.random_tree), 191
LearnerClassifRda
(mlr_learners_classif.rda), 197
LearnerClassifREPTree
(mlr_learners_classif.reptree), 200
LearnerClassifRferns
(mlr_learners_classif.rFerns), 203
LearnerClassifRotationForest
(mlr_learners_classif.rotationForest), 209
LearnerClassifSaeDNN
(mlr_learners_classif.saeDNN), 211
LearnerClassifSda

(mlr_learners_classif.sda), 213
LearnerClassifSdlda (mlr_learners_classif.sdlda), 216
LearnerClassifSGD (mlr_learners_classif.sgd), 218
LearnerClassifSimpleLogistic (mlr_learners_classif.simple_logistic), 221
LearnerClassifSMO (mlr_learners_classif.smo), 224
LearnerClassifSparseLDA (mlr_learners_classif.sparseLDA), 228
LearnerClassifStepPlr (mlr_learners_classif.stepPlr), 230
LearnerClassifTabFM (mlr_learners_classif.tabfm), 232
LearnerClassifTabPFN (mlr_learners_classif.tabpfn), 236
LearnerClassifVotedPerceptron (mlr_learners_classif.voted_perceptron), 240
LearnerCompRisksCoxboost (mlr_learners_cmprsk.coxboost), 243
LearnerCompRisksRandomForestSRC (mlr_learners_cmprsk.rfsrc), 246
LearnerRegrAbess (mlr_learners_regr.abess), 250
LearnerRegrBam (mlr_learners_regr.bam), 256
LearnerRegrBart (mlr_learners_regr.bart), 259
LearnerRegrBcart (mlr_learners_regr.bcart), 262
LearnerRegrBgp (mlr_learners_regr.bgp), 265
LearnerRegrBgpllm (mlr_learners_regr.bgpllm), 267
LearnerRegrBlm (mlr_learners_regr.blm), 270
LearnerRegrBlockForest (mlr_learners_regr.blockforest), 272
LearnerRegrBotorchFullyBayesian (mlr_learners_regr.botorch_fullybayesian), 275
LearnerRegrBotorchMixedSingleTaskGP (mlr_learners_regr.botorch_mixedsingletaskgp), 277
LearnerRegrBotorchSingleTaskGP (mlr_learners_regr.botorch_singletaskgp), 279
LearnerRegrBrnn (mlr_learners_regr.brnn), 281
LearnerRegrBst (mlr_learners_regr.bst), 283
LearnerRegrBtgp (mlr_learners_regr.btgp), 285
LearnerRegrBtgpllm (mlr_learners_regr.btgpllm), 288
LearnerRegrBtlm (mlr_learners_regr.btlm), 291
LearnerRegrCatboost (mlr_learners_regr.catboost), 294
LearnerRegrCForest (mlr_learners_regr.cforest), 298
LearnerRegrCrs (mlr_learners_regr.crs), 302
LearnerRegrCTree (mlr_learners_regr.ctree), 304
LearnerRegrCubist (mlr_learners_regr.cubist), 307
LearnerRegrDecisionStump (mlr_learners_regr.decision_stump), 312
LearnerRegrDecisionTable (mlr_learners_regr.decision_table), 315
LearnerRegrEarth (mlr_learners_regr.earth), 319
LearnerRegrEvtree (mlr_learners_regr.evtree), 322
LearnerRegrExhaustiveSearch (mlr_learners_regr.exhaustive_search), 323
LearnerRegrFNN (mlr_learners_regr.fnn), 326

LearnerRegrFrbs
(mlr_learners_regr.frbs), 328
LearnerRegrGam (mlr_learners_regr.gam),
330
LearnerRegrGAMBoost
(mlr_learners_regr.gamboost),
333
LearnerRegrGaussianProcesses
(mlr_learners_regr.gaussian_processes),
335
LearnerRegrGausspr
(mlr_learners_regr.gausspr),
338
LearnerRegrGBM (mlr_learners_regr.gbm),
340
LearnerRegrGlm (mlr_learners_regr.glm),
343
LearnerRegrGLMBoost
(mlr_learners_regr.glmboost),
345
LearnerRegrGPfit
(mlr_learners_regr.gpfit), 348
LearnerRegrGRF (mlr_learners_regr.grf),
350
LearnerRegrH2ODeeplearning
(mlr_learners_regr.h2o.deeplearning),
353
LearnerRegrH2OGBM
(mlr_learners_regr.h2o.gbm),
356
LearnerRegrH2OGLM
(mlr_learners_regr.h2o.glm),
359
LearnerRegrH2ORandomForest
(mlr_learners_regr.h2o.randomForest),
362
LearnerRegrIBk (mlr_learners_regr.IBk),
365
LearnerRegrKStar
(mlr_learners_regr.kstar), 368
LearnerRegrKSVM
(mlr_learners_regr.ksvm), 371
LearnerRegrLaGP
(mlr_learners_regr.laGP), 374
LearnerRegrLiblineaR
(mlr_learners_regr.liblinear),
376
LearnerRegrLightGBM
(mlr_learners_regr.lightgbm),
379
LearnerRegrLinearRegression
(mlr_learners_regr.linear_regression),
384
LearnerRegrLmer
(mlr_learners_regr.lmer), 387
LearnerRegrM5P (mlr_learners_regr.m5p),
390
LearnerRegrM5Rules
(mlr_learners_regr.M5Rules),
393
LearnerRegrMars
(mlr_learners_regr.mars), 396
LearnerRegrMob (mlr_learners_regr.mob),
398
LearnerRegrMQGam
(mlr_learners_regr.mqgam), 401
LearnerRegrMultilayerPerceptron
(mlr_learners_regr.multilayer_perceptron),
403
LearnerRegrNCVreg
(mlr_learners_regr.cv_ncvreg),
310
LearnerRegrObliqueRandomForest
(mlr_learners_regr.aorsf), 253
LearnerRegrPcr (mlr_learners_regr.pcr),
406
LearnerRegrPenalized
(mlr_learners_regr.penalized),
409
LearnerRegrPlsr
(mlr_learners_regr.plsr), 411
LearnerRegrPoly
(mlr_learners_regr.polyFit),
413
LearnerRegrPolynomial
(mlr_learners_regr.polynomial),
415
LearnerRegrPriorityLasso
(mlr_learners_regr.priority_lasso),
417
LearnerRegrQGam
(mlr_learners_regr.qgam), 421
LearnerRegrRandomForest
(mlr_learners_regr.randomForest),
429
LearnerRegrRandomForestSRC

(mlr_learners_regr.rfsrc), 435
LearnerRegrRandomForestWeka (mlr_learners_regr.random_forest_weka), 423
LearnerRegrRandomTree (mlr_learners_regr.random_tree), 426
LearnerRegrREPTree (mlr_learners_regr.reptree), 432
LearnerRegrRRF (mlr_learners_regr.RRF), 439
LearnerRegrRSM (mlr_learners_regr.rsm), 441
LearnerRegrRVM (mlr_learners_regr.rvm), 444
LearnerRegrSGD (mlr_learners_regr.sgd), 446
LearnerRegrSimpleLinearRegression (mlr_learners_regr.simple_linear_regression), 449
LearnerRegrSMOreg (mlr_learners_regr.smo_reg), 452
LearnerRegrTabFM (mlr_learners_regr.tabfm), 456
LearnerRegrTabPFN (mlr_learners_regr.tabpfn), 460
Learners, *11*, *13*, *16*, *19*, *22*, *25*, *28*, *31*, *34*, *37*, *43*, *47*, *50*, *52*, *55*, *58*, *61*, *63*, *66*, *68*, *71*, *74*, *76*, *81*, *83*, *86*, *88*, *91*, *95*, *98*, *101*, *104*, *107*, *111*, *114*, *117*, *119*, *122*, *125*, *127*, *133*, *136*, *139*, *142*, *144*, *146*, *148*, *151*, *154*, *157*, *160*, *163*, *166*, *168*, *171*, *174*, *177*, *180*, *182*, *184*, *187*, *191*, *194*, *196*, *199*, *202*, *205*, *208*, *210*, *213*, *215*, *217*, *220*, *223*, *227*, *229*, *232*, *236*, *239*, *242*, *245*, *249*, *252*, *255*, *258*, *261*, *264*, *266*, *269*, *271*, *274*, *282*, *285*, *287*, *290*, *293*, *297*, *301*, *304*, *307*, *309*, *312*, *315*, *318*, *321*, *323*, *325*, *327*, *329*, *334*, *337*, *340*, *342*, *345*, *347*, *349*, *352*, *355*, *358*, *361*, *364*, *367*, *370*, *373*, *375*, *378*, *383*, *386*, *389*, *392*, *395*, *397*, *400*, *403*, *405*, *408*, *410*, *413*, *415*, *417*, *420*, *423*, *426*, *429*, *431*, *434*, *438*, *441*, *443*, *445*, *449*, *452*, *455*, *459*, *463*, *466*, *470*, *473*, *476*, *479*, *482*, *485*, *488*, *492*, *495*, *498*, *501*, *507*, *509*, *513*, *517*, *520*, *522*, *525*, *529*, *532*, *536*, *539*, *542*, *546*, *551*
LearnerSurvAorsf (mlr_learners_surv.aorsf), 463
LearnerSurvBlackBoost (mlr_learners_surv.blackboost), 470
LearnerSurvBlockForest (mlr_learners_surv.blockforest), 474
LearnerSurvCForest (mlr_learners_surv.cforest), 477
LearnerSurvCoxboost, *480*, *486*
LearnerSurvCoxboost (mlr_learners_surv.coxboost), 480
LearnerSurvCTree (mlr_learners_surv.ctree), 483
LearnerSurvCVCoxboost, *480*, *486*
LearnerSurvCVCoxboost (mlr_learners_surv.cv_coxboost), 486
LearnerSurvCVGlmnet (mlr_learners_surv.cv_glmnet), 489
LearnerSurvDNN (mlr_learners_surv.survdnn), 537
LearnerSurvFlexReg (mlr_learners_surv.flexreg), 496
LearnerSurvFlexSpline (mlr_learners_surv.flexspline), 499
LearnerSurvGAMBoost (mlr_learners_surv.gamboost), 504
LearnerSurvGamCox (mlr_learners_surv.gam.cox), 502
LearnerSurvGBM (mlr_learners_surv.gbm), 507
LearnerSurvGLMBoost (mlr_learners_surv.glmboost),

510
LearnerSurvGlmnet
(mlr_learners_surv.glmnet), 513
LearnerSurvLearnerSurvBART
(mlr_learners_surv.bart), 467
LearnerSurvMBoost
(mlr_learners_surv.mboost), 518
LearnerSurvNCVsurv
(mlr_learners_surv.cv_ncvsurv), 493
LearnerSurvNelson
(mlr_learners_surv.nelson), 520
LearnerSurvPenalized
(mlr_learners_surv.penalized), 523
LearnerSurvPriorityLasso
(mlr_learners_surv.priority_lasso), 526
LearnerSurvRandomForestSRC
(mlr_learners_surv.rfsrc), 533
LearnerSurvRanger
(mlr_learners_surv.ranger), 530
LearnerSurvSVM (mlr_learners_surv.svm), 540
LearnerSurvXgboostAFT
(mlr_learners_surv.xgboost.aft), 543
LearnerSurvXgboostCox
(mlr_learners_surv.xgboost.cox), 547
list_mlr3learners, 8
lrn(), *9*, *12*, *14*, *17*, *20*, *24*, *26*, *30*, *32*, *35*, *38*, *39*, *44*, *48*, *51*, *53*, *56*, *59*, *62*, *65*, *69*, *72*, *75*, *77*, *79*, *82*, *84*, *87*, *89*, *92*, *96*, *99*, *102*, *105*, *108*, *112*, *115*, *118*, *121*, *123*, *126*, *128*, *134*, *137*, *140*, *143*, *145*, *147*, *149*, *152*, *155*, *158*, *161*, *164*, *167*, *169*, *173*, *175*, *178*, *181*, *183*, *185*, *189*, *192*, *195*, *197*, *200*, *203*, *206*, *209*, *211*, *214*, *216*, *218*, *222*, *225*, *228*, *230*, *233*, *237*, *240*, *243*, *246*, *250*, *253*, *257*, *259*, *262*, *265*, *267*, *270*, *273*, *275*, *277*, *279*, *281*, *283*, *286*, *289*, *291*, *294*, *298*, *302*, *305*, *308*, *310*, *313*, *316*, *319*, *324*, *326*, *328*, *331*, *333*, *335*, *338*, *341*, *343*, *346*, *348*, *350*, *353*, *356*, *359*, *362*, *365*, *368*, *371*, *374*, *376*, *379*, *384*, *387*, *390*, *393*, *396*, *398*, *401*, *403*, *406*, *409*, *411*, *414*, *415*, *418*, *421*, *424*, *427*, *430*, *433*, *435*, *439*, *442*, *444*, *447*, *450*, *453*, *457*, *461*, *464*, *468*, *471*, *474*, *477*, *481*, *483*, *486*, *489*, *494*, *496*, *499*, *502*, *505*, *507*, *510*, *515*, *518*, *521*, *523*, *526*, *530*, *533*, *538*, *540*, *543*, *548*

marshal_model(), *277*, *280*
mlr3::Learner, *10*, *12*, *15*, *18*, *21*, *25*, *27*, *30*, *33*, *36*, *39*, *42*, *46*, *49*, *51*, *54*, *56*, *60*, *63*, *66*, *67*, *70*, *73*, *75*, *78*, *80*, *83*, *85*, *88*, *91*, *94*, *97*, *100*, *103*, *106*, *110*, *113*, *116*, *119*, *121*, *124*, *127*, *131*, *132*, *135*, *138*, *141*, *143*, *146*, *148*, *150*, *153*, *156*, *159*, *162*, *165*, *168*, *170*, *173*, *176*, *179*, *181*, *184*, *186*, *189*, *193*, *195*, *198*, *201*, *204*, *207*, *210*, *212*, *214*, *216*, *219*, *222*, *226*, *229*, *231*, *234*, *238*, *241*, *244*, *248*, *252*, *254*, *258*, *260*, *263*, *266*, *268*, *271*, *273*, *276*, *278*, *279*, *281*, *284*, *287*, *289*, *292*, *296*, *300*, *303*, *306*, *308*, *311*, *313*, *317*, *320*, *322*, *324*, *326*, *329*, *332*, *334*, *336*, *339*, *341*, *344*, *346*, *349*, *351*, *355*, *358*, *361*, *363*, *366*, *369*, *372*, *374*, *377*, *382*, *385*, *388*, *391*, *394*, *397*, *399*, *402*, *404*, *407*, *410*, *412*, *414*, *416*, *419*, *422*, *424*, *428*, *430*, *433*, *437*, *440*, *442*, *445*, *447*, *450*, *454*, *458*, *462*, *465*, *469*, *472*, *475*, *479*, *481*, *484*, *487*, *491*, *494*, *497*, *500*, *503*, *506*, *508*, *511*, *516*, *519*, *521*, *524*, *528*, *531*, *535*, *538*, *541*, *545*, *549*
mlr3::LearnerClassif, *10*, *12*, *15*, *18*, *21*, *25*, *27*, *30*, *33*, *36*, *39*, *42*, *46*, *49*, *51*, *54*, *56*, *60*, *63*, *66*, *67*, *70*, *73*, *75*, *78*, *80*, *83*, *85*, *88*, *91*, *94*, *97*, *100*, *103*, *106*, *110*, *113*, *116*, *119*, *121*, *124*, *127*, *132*, *135*, *138*, *141*, *143*, *146*, *148*, *150*, *153*, *156*, *159*, *162*, *165*, *168*, *170*, *173*, *176*, *179*, *181*, *184*, *186*, *189*, *193*, *195*, *198*, *201*, *204*, *207*, *210*, *212*, *214*, *216*, *219*, *222*, *226*, *229*, *231*, *234*, *238*, *241*
mlr3::LearnerRegr, *252*, *254*, *258*, *260*, *263*,

266, 268, 271, 273, 276, 278, 279, 281, 284, 287, 289, 292, 296, 300, 303, 306, 308, 311, 313, 317, 320, 322, 324, 326, 329, 332, 334, 336, 339, 341, 344, 346, 349, 351, 355, 358, 361, 363, 366, 369, 372, 374, 377, 382, 385, 388, 391, 394, 397, 399, 402, 404, 407, 410, 412, 414, 416, 419, 422, 424, 428, 430, 433, 437, 440, 442, 445, 447, 450, 454, 458, 462
mlr3::marshal_model(), 18, 31, 57, 61, 73, 74, 107, 113, 116, 122, 135, 138, 157, 160, 163, 171, 177, 190, 193, 201, 220, 223, 227, 235, 239, 241, 278, 314, 318, 337, 367, 370, 385, 391, 394, 405, 425, 428, 434, 448, 451, 455, 459, 462, 539
mlr3::Measure, 131, 382
mlr3::mlr_learners, 11, 13, 16, 19, 22, 25, 28, 31, 34, 37, 43, 47, 50, 52, 55, 58, 61, 63, 66, 68, 71, 74, 76, 81, 83, 86, 88, 91, 95, 98, 101, 104, 107, 111, 114, 117, 119, 122, 125, 127, 133, 136, 139, 142, 144, 146, 148, 151, 154, 157, 160, 163, 166, 168, 171, 174, 177, 180, 182, 184, 187, 191, 194, 196, 199, 202, 205, 208, 210, 213, 215, 217, 220, 223, 227, 229, 232, 236, 239, 242, 245, 249, 252, 255, 258, 261, 264, 266, 269, 271, 274, 282, 285, 287, 290, 293, 297, 301, 304, 307, 309, 312, 315, 318, 321, 323, 325, 327, 329, 334, 337, 340, 342, 345, 347, 349, 352, 355, 358, 361, 364, 367, 370, 373, 375, 378, 383, 386, 389, 392, 395, 397, 400, 403, 405, 408, 410, 413, 415, 417, 420, 423, 426, 429, 431, 434, 438, 441, 443, 445, 449, 452, 455, 459, 463, 466, 470, 473, 476, 479, 482, 485, 488, 492, 495, 498, 501, 507, 509, 513, 517, 520, 522, 525, 529, 532, 536, 539, 542, 546, 551
mlr3::Resampling, 543, 549
mlr3::set_threads(), 23, 257
mlr3::Task, 543, 549
mlr3::unmarshal_model(), 19, 31, 57, 61, 107, 114, 117, 122, 136, 139, 157, 160, 163, 171, 177, 190, 193, 202, 220, 223, 227, 235, 239, 241, 278, 314, 318, 337, 367, 370, 386, 391, 395, 405, 425, 428, 434, 448, 451, 455, 459, 462, 539
mlr3cmprsk::LearnerCompRisks, 244, 248
mlr3extralearners (mlr3extralearners-package), 6
mlr3extralearners-package, 6
mlr3proba::LearnerSurv, 465, 469, 472, 475, 479, 481, 484, 487, 491, 494, 497, 500, 503, 506, 508, 511, 516, 519, 521, 524, 528, 531, 535, 538, 541, 545, 549
mlr_learners, 8
mlr_learners_classif.abess, 9
mlr_learners_classif.ada, 11
mlr_learners_classif.adabag, 14
mlr_learners_classif.AdaBoostM1, 17
mlr_learners_classif.aorsf, 20
mlr_learners_classif.bam, 23
mlr_learners_classif.bart, 26
mlr_learners_classif.bayes_net, 29
mlr_learners_classif.blockforest, 32
mlr_learners_classif.bst, 35
mlr_learners_classif.C50, 38
mlr_learners_classif.catboost, 39
mlr_learners_classif.cforest, 44
mlr_learners_classif.ctree, 47
mlr_learners_classif.cv_ncvreg, 50
mlr_learners_classif.dbnDNN, 53
mlr_learners_classif.decision_stump, 55
mlr_learners_classif.decision_table, 58
mlr_learners_classif.dlda, 62
mlr_learners_classif.earth, 64
mlr_learners_classif.evtree, 67
mlr_learners_classif.exhaustive_search, 69
mlr_learners_classif.fastai, 71
mlr_learners_classif.fnn, 75
mlr_learners_classif.gam, 77
mlr_learners_classif.gamboost, 79
mlr_learners_classif.gausspr, 82
mlr_learners_classif.gbm, 84
mlr_learners_classif.glmboost, 87

mlr_learners_classif.glmer, 89
mlr_learners_classif.h2o.deeplearning, 92
mlr_learners_classif.h2o.gbm, 95
mlr_learners_classif.h2o.glm, 98
mlr_learners_classif.h2o.randomForest, 102
mlr_learners_classif.IBk, 105
mlr_learners_classif.imbalanced_rfsrc, 108
mlr_learners_classif.J48, 112
mlr_learners_classif.JRip, 115
mlr_learners_classif.knn, 118
mlr_learners_classif.kstar, 120
mlr_learners_classif.ksvm, 123
mlr_learners_classif.liblinear, 125
mlr_learners_classif.lightgbm, 128
mlr_learners_classif.LMT, 134
mlr_learners_classif.log_reg, *343*
mlr_learners_classif.logistic, 137
mlr_learners_classif.lssvm, 140
mlr_learners_classif.lvq1, 142
mlr_learners_classif.mda, 145
mlr_learners_classif.mdeb, 147
mlr_learners_classif.mlp_rsnns, 149
mlr_learners_classif.mob, 152
mlr_learners_classif.multilayer_perceptron, 155
mlr_learners_classif.naive_bayes_multinomial, 158
mlr_learners_classif.naive_bayes_weka, 161
mlr_learners_classif.neuralnet, 164
mlr_learners_classif.nnTrain, 167
mlr_learners_classif.OneR, 169
mlr_learners_classif.opls, 172
mlr_learners_classif.PART, 175
mlr_learners_classif.penalized, 178
mlr_learners_classif.plsdaCaret, 181
mlr_learners_classif.polyFit, 183
mlr_learners_classif.priority_lasso, 184
mlr_learners_classif.random_forest_weka, 188
mlr_learners_classif.random_tree, 191
mlr_learners_classif.randomForest, 194
mlr_learners_classif.rda, 197
mlr_learners_classif.reptree, 200
mlr_learners_classif.rFerns, 203
mlr_learners_classif.rfsrc, 205
mlr_learners_classif.rotationForest, 209
mlr_learners_classif.saeDNN, 211
mlr_learners_classif.sda, 213
mlr_learners_classif.sdlda, 216
mlr_learners_classif.sgd, 218
mlr_learners_classif.simple_logistic, 221
mlr_learners_classif.smo, 224
mlr_learners_classif.sparseLDA, 228
mlr_learners_classif.stepPlr, 230
mlr_learners_classif.tabfm, 232
mlr_learners_classif.tabpfn, 236
mlr_learners_classif.voted_perceptron, 240
mlr_learners_cmprsk.coxboost, 243
mlr_learners_cmprsk.rfsrc, 246
mlr_learners_regr.abess, 250
mlr_learners_regr.aorsf, 253
mlr_learners_regr.bam, 256
mlr_learners_regr.bart, 259
mlr_learners_regr.bcart, 262
mlr_learners_regr.bgp, 265
mlr_learners_regr.bgpllm, 267
mlr_learners_regr.blm, 270
mlr_learners_regr.blockforest, 272
mlr_learners_regr.botorch_fullybayesian, 275
mlr_learners_regr.botorch_mixedsingletaskgp, 277
mlr_learners_regr.botorch_singletaskgp, 279
mlr_learners_regr.brnn, 281
mlr_learners_regr.bst, 283
mlr_learners_regr.btgp, 285
mlr_learners_regr.btgpllm, 288
mlr_learners_regr.btlm, 291
mlr_learners_regr.catboost, 294
mlr_learners_regr.cforest, 298
mlr_learners_regr.crs, 302
mlr_learners_regr.ctree, 304
mlr_learners_regr.cubist, 307
mlr_learners_regr.cv_ncvreg, 310
mlr_learners_regr.decision_stump, 312
mlr_learners_regr.decision_table, 315
mlr_learners_regr.earth, 319

mlr_learners_regr.evtree, 322
mlr_learners_regr.exhaustive_search, 323
mlr_learners_regr.fnn, 326
mlr_learners_regr.frbs, 328
mlr_learners_regr.gam, 330
mlr_learners_regr.gamboost, 333
mlr_learners_regr.gaussian_processes, 335
mlr_learners_regr.gausspr, 338
mlr_learners_regr.gbm, 340
mlr_learners_regr.glm, 343
mlr_learners_regr.glmboost, 345
mlr_learners_regr.gpfit, 348
mlr_learners_regr.grf, 350
mlr_learners_regr.h2o.deeplearning, 353
mlr_learners_regr.h2o.gbm, 356
mlr_learners_regr.h2o.glm, 359
mlr_learners_regr.h2o.randomForest, 362
mlr_learners_regr.IBk, 365
mlr_learners_regr.kstar, 368
mlr_learners_regr.ksvm, 371
mlr_learners_regr.laGP, 374
mlr_learners_regr.liblinear, 376
mlr_learners_regr.lightgbm, 379
mlr_learners_regr.linear_regression, 384
mlr_learners_regr.lmer, 387
mlr_learners_regr.m5p, 390
mlr_learners_regr.M5Rules, 393
mlr_learners_regr.mars, 396
mlr_learners_regr.mob, 398
mlr_learners_regr.mqgam, 401
mlr_learners_regr.multilayer_perceptron, 403
mlr_learners_regr.pcr, 406
mlr_learners_regr.penalized, 409
mlr_learners_regr.plsr, 411
mlr_learners_regr.polyFit, 413
mlr_learners_regr.polynomial, 415
mlr_learners_regr.priority_lasso, 417
mlr_learners_regr.qgam, 421
mlr_learners_regr.random_forest_weka, 423
mlr_learners_regr.random_tree, 426
mlr_learners_regr.randomForest, 429
mlr_learners_regr.reptree, 432
mlr_learners_regr.rfsrc, 435
mlr_learners_regr.RRF, 439
mlr_learners_regr.rsm, 441
mlr_learners_regr.rvm, 444
mlr_learners_regr.sgd, 446
mlr_learners_regr.simple_linear_regression, 449
mlr_learners_regr.smo_reg, 452
mlr_learners_regr.tabfm, 456
mlr_learners_regr.tabpfn, 460
mlr_learners_surv.aorsf, 463
mlr_learners_surv.bart, 467
mlr_learners_surv.blackboost, 470
mlr_learners_surv.blockforest, 474
mlr_learners_surv.cforest, 477
mlr_learners_surv.coxboost, 480
mlr_learners_surv.ctree, 483
mlr_learners_surv.cv_coxboost, 486
mlr_learners_surv.cv_glmnet, 489
mlr_learners_surv.cv_glmnet(), *514*
mlr_learners_surv.cv_ncvsurv, 493
mlr_learners_surv.flexreg, 496
mlr_learners_surv.flexspline, 499
mlr_learners_surv.gam.cox, 502
mlr_learners_surv.gamboost, 504
mlr_learners_surv.gbm, 507
mlr_learners_surv.glmboost, 510
mlr_learners_surv.glmnet, 513
mlr_learners_surv.mboost, 518
mlr_learners_surv.nelson, 520
mlr_learners_surv.penalized, 523
mlr_learners_surv.priority_lasso, 526
mlr_learners_surv.ranger, 530
mlr_learners_surv.rfsrc, 533
mlr_learners_surv.survdnn, 537
mlr_learners_surv.svm, 540
mlr_learners_surv.xgboost.aft, 543
mlr_learners_surv.xgboost.cox, 547

PredictionSurv, *467*

R6, *10*, *13*, *15*, *18*, *21*, *25*, *27*, *31*, *33*, *36*, *39*, *46*, *49*, *52*, *54*, *57*, *60*, *63*, *66*, *67*, *70*, *73*, *75*, *78*, *83*, *85*, *91*, *94*, *97*, *100*, *103*, *106*, *110*, *113*, *116*, *119*, *121*, *124*, *127*, *132*, *135*, *138*, *141*, *143*, *146*, *148*, *150*, *153*, *156*, *159*, *162*, *165*, *168*, *170*, *174*, *176*, *179*, *181*,

184, *186*, *190*, *193*, *196*, *198*, *201*, *204*, *207*, *210*, *212*, *214*, *216*, *219*, *223*, *226*, *229*, *231*, *235*, *238*, *241*, *244*, *248*, *252*, *255*, *258*, *260*, *263*, *266*, *268*, *271*, *274*, *276*, *278*, *280*, *282*, *284*, *287*, *290*, *292*, *300*, *303*, *306*, *308*, *311*, *314*, *317*, *321*, *322*, *324*, *326*, *329*, *332*, *336*, *339*, *342*, *344*, *349*, *351*, *355*, *358*, *361*, *363*, *366*, *369*, *372*, *375*, *377*, *382*, *385*, *388*, *391*, *394*, *397*, *400*, *402*, *405*, *407*, *410*, *412*, *414*, *416*, *419*, *422*, *425*, *428*, *431*, *434*, *437*, *440*, *442*, *445*, *448*, *451*, *454*, *458*, *462*, *465*, *469*, *472*, *475*, *479*, *481*, *485*, *488*, *491*, *495*, *497*, *500*, *504*, *506*, *509*, *512*, *516*, *519*, *521*, *524*, *528*, *532*, *535*, *539*, *541*, *546*, *550*

regr.gam, *256*
regr.polyFit, *415*
RSNNS::mlp(), *149*

Task, *401*, *421*
task, *89*, *387*

unmarshal_model(), *277*, *280*
utils::install.packages, *8*