Package 'mlr3learners'

September 8, 2026

Title Recommended Learners for 'mlr3'

Version 0.16.0

Description Recommended Learners for 'mlr3'. Extends 'mlr3' with interfaces to essential machine learning packages on CRAN. This includes, but is not limited to: (penalized) linear and logistic regression, linear and quadratic discriminant analysis, k-nearest neighbors, naive Bayes, support vector machines, and gradient boosting.

License LGPL-3

URL https://mlr3learners.mlr-org.com, https://github.com/mlr-org/mlr3learners

BugReports https://github.com/mlr-org/mlr3learners/issues

Depends mlr3 (>= 1.8.0), R (>= 3.4.0)

Imports checkmate, data.table, methods, mlr3misc (>= 0.9.4), paradox (>= 1.0.0), R6

Suggests DiceKriging, e1071, future, glmnet, kknn, knitr, lgr, MASS, mirai, nnet, pracma, ranger, rgenoud, rmarkdown, testthat (>= 3.0.0), xgboost (>= 3.2.0.1)

Config/testthat/edition 3

Encoding UTF-8

NeedsCompilation yes

Collate 'aaa.R' 'LearnerClassifCVGlmnet.R' 'LearnerClassifGlmnet.R' 'LearnerClassifKKNN.R' 'LearnerClassifLDA.R' 'LearnerClassifLogReg.R' 'LearnerClassifMultinom.R' 'LearnerClassifNaiveBayes.R' 'LearnerClassifNnet.R' 'LearnerClassifQDA.R' 'LearnerClassifRanger.R' 'LearnerClassifSVM.R' 'LearnerClassifXgboost.R' 'LearnerRegrCVGlmnet.R' 'LearnerRegrGlmnet.R' 'LearnerRegrKKNN.R' 'LearnerRegrKM.R' 'LearnerRegrLM.R' 'LearnerRegrNnet.R' 'LearnerRegrRanger.R' 'LearnerRegrSVM.R' 'LearnerRegrXgboost.R' 'bibentries.R' 'helpers.R' 'helpers_glmnet.R' 'helpers_ranger.R' 'helpers_xgboost.R' 'zzz.R'

Config/roxygen2/version 8.1.0

Author Michel Lang [aut] (ORCID: <https://orcid.org/0000-0001-9754-0393>),
Quay Au [aut] (ORCID: <https://orcid.org/0000-0002-5252-8902>),
Stefan Coors [aut] (ORCID: <https://orcid.org/0000-0002-7465-2146>),
Patrick Schratz [aut] (ORCID: <https://orcid.org/0000-0003-0748-6624>),
Marc Becker [cre, aut] (ORCID: <https://orcid.org/0000-0002-8115-0400>),
John Zobolas [aut] (ORCID: <https://orcid.org/0000-0002-3609-8674>),
Alexander Winterstetter [ctb],
Toby Hocking [ctb] (ORCID: <https://orcid.org/0000-0002-3146-0865>)

Maintainer Marc Becker <marcbecker@posteo.de>

Repository CRAN

Date/Publication 2026-09-08 13:00:02 UTC

Contents

mlr3learners-package . . . 3
mlr_learners_classif.cv_glmnet . . . 4
mlr_learners_classif.glmnet . . . 7
mlr_learners_classif.kknn . . . 11
mlr_learners_classif.lda . . . 14
mlr_learners_classif.log_reg . . . 17
mlr_learners_classif.multinom . . . 20
mlr_learners_classif.naive_bayes . . . 22
mlr_learners_classif.nnet . . . 24
mlr_learners_classif.qda . . . 27
mlr_learners_classif.ranger . . . 30
mlr_learners_classif.svm . . . 33
mlr_learners_classif.xgboost . . . 36
mlr_learners_regr.cv_glmnet . . . 41
mlr_learners_regr.glmnet . . . 45
mlr_learners_regr.kknn . . . 49
mlr_learners_regr.km . . . 52
mlr_learners_regr.lm . . . 55
mlr_learners_regr.nnet . . . 58
mlr_learners_regr.ranger . . . 60
mlr_learners_regr.svm . . . 65
mlr_learners_regr.xgboost . . . 67

Index **73**

mlr3learners-package *mlr3learners: Recommended Learners for 'mlr3'*

Description

More learners are implemented in the mlr3extralearners package. A guide on how to create custom learners is covered in the book: https://mlr3book.mlr-org.com. Feel invited to contribute a missing learner to the **mlr3** ecosystem!

Author(s)

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

Authors:

- Marc Becker <marcbecker@posteo.de> (ORCID)
- Michel Lang <michellang@gmail.com> (ORCID)
- Quay Au <quayau@gmail.com> (ORCID)
- Stefan Coors <mail@stefancoors.de> (ORCID)
- Patrick Schratz <patrick.schratz@gmail.com> (ORCID)
- John Zobolas <bblodfon@gmail.com> (ORCID)

Other contributors:

- Alexander Winterstetter <alexanderwinterstetter@gmail.com> [contributor]
- Toby Hocking <toby.hocking@r-project.org> (ORCID) [contributor]

See Also

Useful links:

- https://mlr3learners.mlr-org.com
- https://github.com/mlr-org/mlr3learners
- Report bugs at https://github.com/mlr-org/mlr3learners/issues

`mlr_learners_classif.cv_glmnet` *GLM with Elastic Net Regularization Classification Learner*

Description

Generalized linear models with elastic net regularization. Calls `glmnet::cv.glmnet()` from package **glmnet**.

The default for hyperparameter `family` is set to `"binomial"` or `"multinomial"`, depending on the number of classes.

Custom mlr3 parameters

- `seed`:
  - Optional integer used to seed the call to `glmnet::cv.glmnet()`, making its random fold assignment, and therefore the selected lambda, reproducible.
  - The global random state is reset afterwards, so it is left unchanged.
  - Defaults to `NA`, in which case no seed is set and the global random state is used.

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.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.cv_glmnet")
lrn("classif.cv_glmnet")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3learners**, **glmnet**

Parameters

Id	Type	Default	Levels	Range
lambda	untyped	NULL		-
type.measure	character	deviance	deviance, class, auc, mse, mae	-
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		-
type.logistic	character	-	Newton, modified.Newton	-
type.multinomial	character	-	ungrouped, grouped	-
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		$(-\infty, \infty)$
pmax	integer	NULL		$(-\infty, \infty)$
s	numeric	lambda.1se		$[0, \infty)$
predict.gamma	numeric	gamma.1se		$[0, 1]$
exact	logical	FALSE	TRUE, FALSE	-
use_pred_offset	logical	-	TRUE, FALSE	-
seed	integer	-		$(-\infty, \infty)$

Internal Encoding

Starting with **mlr3** v0.5.0, the order of class labels is reversed prior to model fitting to comply to the `stats::glm()` convention that the negative class is provided as the first factor level.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifCVGlmnet`

Methods

Public methods:

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

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

Usage:

```
LearnerClassifCVGlmnet$new()
```

`LearnerClassifCVGlmnet$selected_features()`: Returns the set of selected features as reported by `glmnet::predict.glmnet()` with `type` set to `"nonzero"`.

Usage:

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

Arguments:

`lambda` (`numeric(1)`)
Custom `lambda`, defaults to the active lambda depending on parameter set.

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

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

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.cv_glmnet")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_classif.glmnet`

GLM with Elastic Net Regularization Classification Learner

Description

Generalized linear models with elastic net regularization. Calls `glmnet::glmnet()` from package **glmnet**.

The default for hyperparameter `family` is set to `"binomial"` or `"multinomial"`, depending on the number of classes.

Details

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 an 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 counterparts in `mlr_learners_classif.cv_glmnet()` or `mlr_learners_regr.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).

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.glmnet")
lrn("classif.glmnet")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3learners**, **glmnet**

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		-
type.logistic	character	-	Newton, modified.Newton	-
type.multinomial	character	-	ungrouped, grouped	-
relax	logical	FALSE	TRUE, FALSE	-
trace.it	integer	0		$[0, 1]$
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		$[1, \infty)$
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		$[0, 1]$
use_pred_offset	logical	-	TRUE, FALSE	-

Internal Encoding

Starting with **mlr3** v0.5.0, the order of class labels is reversed prior to model fitting to comply to the `stats::glm()` convention that the negative class is provided as the first factor level.

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.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifGlmnet`

Methods

Public methods:

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

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

Usage:

```
LearnerClassifGlmnet$new()
```

`LearnerClassifGlmnet$selected_features()`: Returns the set of selected features as reported by `glmnet::predict.glmnet()` with `type` set to `"nonzero"`.

Usage:

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

Arguments:

`lambda` (`numeric(1)`)
Custom `lambda`, defaults to the active lambda depending on parameter set.

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

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

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: mlr_learners_classif.cv_glmnet, mlr_learners_classif.kknn, mlr_learners_classif.lda, mlr_learners_classif.log_reg, mlr_learners_classif.multinom, mlr_learners_classif.naive_bayes, mlr_learners_classif.nnet, mlr_learners_classif.qda, mlr_learners_classif.ranger, mlr_learners_classif.svm, mlr_learners_classif.xgboost, mlr_learners_regr.cv_glmnet, mlr_learners_regr.glmnet, mlr_learners_regr.kknn, mlr_learners_regr.km, mlr_learners_regr.lm, mlr_learners_regr.nnet, mlr_learners_regr.ranger, mlr_learners_regr.svm, mlr_learners_regr.xgboost

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.glmnet")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

---

mlr_learners_classif.kknn

k-Nearest-Neighbor Classification Learner

Description

k-Nearest-Neighbor classification. Calls kknn::kknn() from package **kknn**.

Initial parameter values

- store_model:
  - See note.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.kknn")
lrn("classif.kknn")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3learners**, **kknn**

Parameters

Id	Type	Default	Levels
k	integer	7	
distance	numeric	2	
kernel	character	optimal	rectangular, triangular, epanechnikov, biweight, triweight, cos, inv, gaussian, rank, optim
scale	logical	TRUE	TRUE, FALSE
ykernel	untyped	NULL	
store_model	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifKKNN`

Methods

Public methods:

- `LearnerClassifKKNN$new()`
- `LearnerClassifKKNN$clone()`

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

Usage:

```
LearnerClassifKKNN$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

There is no training step for k-NN models, just storing the training data to process it during the predict step. Therefore, `$model` returns a list with the following elements:

- `formula`: Formula for calling `kknn::kknn()` during `$predict()`.
- `data`: Training data for calling `kknn::kknn()` during `$predict()`.
- `pv`: Training parameters for calling `kknn::kknn()` during `$predict()`.
- `kknn`: Model as returned by `kknn::kknn()`, only available **after** `$predict()` has been called. This is not stored by default, you must set hyperparameter `store_model` to `TRUE`.

References

Hechenbichler, Klaus, Schliep, Klaus (2004). "Weighted k-nearest-neighbor techniques and ordinal classification." Technical Report Discussion Paper 399, SFB 386, Ludwig-Maximilians University Munich. doi:10.5282/ubm/epub.1769.

Samworth, J R (2012). "Optimal weighted nearest neighbour classifiers." *The Annals of Statistics*, **40**(5), 2733–2763. doi:10.1214/12AOS1049.

Cover, Thomas, Hart, Peter (1967). "Nearest neighbor pattern classification." *IEEE transactions on information theory*, **13**(1), 21–27. doi:10.1109/TIT.1967.1053964.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.kknn")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_classif.lda`

Linear Discriminant Analysis Classification Learner

Description

Linear discriminant analysis. Calls `MASS::lda()` from package **MASS**.

Details

Parameters `method` and `prior` exist for training and prediction but accept different values for each. Therefore, arguments for the predict stage have been renamed to `predict.method` and `predict.prior`, respectively.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.lda")
lrn("classif.lda")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3learners**, **MASS**

Parameters

Id	Type	Default	Levels	Range
dimen	untyped	-		-
method	character	moment	moment, mle, mve, t	-
nu	integer	-		$(-\infty, \infty)$
predict.method	character	plug-in	plug-in, predictive, debiased	-
predict.prior	untyped	-		-
prior	untyped	-		-
tol	numeric	-		$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifLDA`

Methods

Public methods:

- `LearnerClassifLDA$new()`
- `LearnerClassifLDA$clone()`

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

Usage:

```
LearnerClassifLDA$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

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

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: mlr_learners_classif.cv_glmnet, mlr_learners_classif.glmnet, mlr_learners_classif.kknn, mlr_learners_classif.log_reg, mlr_learners_classif.multinom, mlr_learners_classif.naive_bayes, mlr_learners_classif.nnet, mlr_learners_classif.qda, mlr_learners_classif.ranger, mlr_learners_classif.svm, mlr_learners_classif.xgboost, mlr_learners_regr.cv_glmnet, mlr_learners_regr.glmnet, mlr_learners_regr.kknn, mlr_learners_regr.km, mlr_learners_regr.lm, mlr_learners_regr.nnet, mlr_learners_regr.ranger, mlr_learners_regr.svm, mlr_learners_regr.xgboost

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.lda")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_classif.log_reg`
Logistic Regression Classification Learner

Description

Classification via logistic regression. Calls `stats::glm()` with `family` set to `binomial(link = <link>)` with `link` either as `"logit"` (default) or `"probit"`.

Internal Encoding

Starting with **mlr3** v0.5.0, the order of class labels is reversed prior to model fitting to comply to the `stats::glm()` convention that the negative class is provided as the first factor level.

Initial parameter values

- `model`:
  - Actual default: `TRUE`.
  - Adjusted default: `FALSE`.
  - Reason for change: Save some memory.

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.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.log_reg")
lrn("classif.log_reg")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3learners**, ’stats’

Parameters

Id	Type	Default	Levels	Range
dispersion	untyped	NULL		-
epsilon	numeric	1e-08		$(-\infty, \infty)$
etastart	untyped	-		-
link	character	-	logit, probit	-
maxit	numeric	25		$(-\infty, \infty)$
model	logical	TRUE	TRUE, FALSE	-
mustart	untyped	-		-
singular.ok	logical	TRUE	TRUE, FALSE	-
start	untyped	NULL		-
trace	logical	FALSE	TRUE, FALSE	-
x	logical	FALSE	TRUE, FALSE	-
y	logical	TRUE	TRUE, FALSE	-
use_pred_offset	logical	-	TRUE, FALSE	-

Contrasts

To ensure reproducibility, this learner always uses the default contrasts:

- `contr.treatment()` for unordered factors, and
- `contr.poly()` for ordered factors.

Setting the option `"contrasts"` does not have any effect. Instead, set the respective hyperparameter or use **mlr3pipelines** to create dummy features.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifLogReg`

Methods

Public methods:

- `LearnerClassifLogReg$new()`
- `LearnerClassifLogReg$clone()`

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

Usage:

```
LearnerClassifLogReg$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.log_reg")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_classif.multinom`
Multinomial log-linear learner via neural networks

Description

Multinomial log-linear models via neural networks. Calls `nnet::multinom()` from package **nnet**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.multinom")
lrn("classif.multinom")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3learners**, **nnet**

Parameters

Id	Type	Default	Levels	Range
Hess	logical	FALSE	TRUE, FALSE	-
abstol	numeric	1e-04		$(-\infty, \infty)$
censored	logical	FALSE	TRUE, FALSE	-
decay	numeric	0		$(-\infty, \infty)$
entropy	logical	FALSE	TRUE, FALSE	-
mask	untyped	-		-
maxit	integer	100		$[1, \infty)$
MaxNWts	integer	1000		$[1, \infty)$
model	logical	FALSE	TRUE, FALSE	-
linout	logical	FALSE	TRUE, FALSE	-
rang	numeric	0.7		$(-\infty, \infty)$
reltol	numeric	1e-08		$(-\infty, \infty)$
size	integer	-		$[1, \infty)$
skip	logical	FALSE	TRUE, FALSE	-
softmax	logical	FALSE	TRUE, FALSE	-
summ	character	0	0, 1, 2, 3	-
trace	logical	TRUE	TRUE, FALSE	-
Wts	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifMultinom`

Methods

Public methods:

- `LearnerClassifMultinom$new()`
- `LearnerClassifMultinom$clone()`

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

Usage:

```
LearnerClassifMultinom$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.multinom")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

---

mlr_learners_classif.naive_bayes

Naive Bayes Classification Learner

Description

Naive Bayes classification. Calls `e1071::naiveBayes()` from package **e1071**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.naive_bayes")
lrn("classif.naive_bayes")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor"
- Required Packages: **mlr3**, **mlr3learners**, **e1071**

Parameters

Id	Type	Default	Range
eps	numeric	0	$(-\infty, \infty)$
laplace	numeric	0	$[0, \infty)$
threshold	numeric	0.001	$(-\infty, \infty)$

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifNaiveBayes`

Methods

Public methods:

- `LearnerClassifNaiveBayes$new()`
- `LearnerClassifNaiveBayes$clone()`

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

Usage:

```
LearnerClassifNaiveBayes$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.

- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: mlr_learners_classif.cv_glmnet, mlr_learners_classif.glmnet, mlr_learners_classif.kknn, mlr_learners_classif.lda, mlr_learners_classif.log_reg, mlr_learners_classif.multinom, mlr_learners_classif.nnet, mlr_learners_classif.qda, mlr_learners_classif.ranger, mlr_learners_classif.svm, mlr_learners_classif.xgboost, mlr_learners_regr.cv_glmnet, mlr_learners_regr.glmnet, mlr_learners_regr.kknn, mlr_learners_regr.km, mlr_learners_regr.lm, mlr_learners_regr.nnet, mlr_learners_regr.ranger, mlr_learners_regr.svm, mlr_learners_regr.xgboost

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.naive_bayes")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

---

mlr_learners_classif.nnet

Classification Neural Network Learner

Description

Single Layer Neural Network. Calls nnet::nnet.formula() from package **nnet**.

Note that modern neural networks with multiple layers are connected via package mlr3torch.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.nnet")
lrn("classif.nnet")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3learners**, **nnet**

Parameters

Id	Type	Default	Levels	Range
Hess	logical	FALSE	TRUE, FALSE	-
MaxNWts	integer	1000		$[1, \infty)$
Wts	untyped	-		-
abstol	numeric	1e-04		$(-\infty, \infty)$
censored	logical	FALSE	TRUE, FALSE	-
contrasts	untyped	NULL		-
decay	numeric	0		$(-\infty, \infty)$
mask	untyped	-		-
maxit	integer	100		$[1, \infty)$
na.action	untyped	-		-
rang	numeric	0.7		$(-\infty, \infty)$
reltol	numeric	1e-08		$(-\infty, \infty)$
size	integer	3		$[0, \infty)$
skip	logical	FALSE	TRUE, FALSE	-
subset	untyped	-		-
trace	logical	TRUE	TRUE, FALSE	-
formula	untyped	-		-

Initial parameter values

- `size`:
  - Adjusted default: 3L.
  - Reason for change: no default in `nnet()`.

Custom mlr3 parameters

- `formula`: if not provided, the formula is set to `task$formula()`.

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifNnet`

Methods

Public methods:

- `LearnerClassifNnet$new()`
- `LearnerClassifNnet$clone()`

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

Usage:

```
LearnerClassifNnet$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Ripley BD (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press. doi:10.1017/cbo9780511812651.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.nnet")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

---

mlr_learners_classif.qda

Quadratic Discriminant Analysis Classification Learner

Description

Quadratic discriminant analysis. Calls `MASS::qda()` from package **MASS**.

Details

Parameters `method` and `prior` exist for training and prediction but accept different values for each. Therefore, arguments for the predict stage have been renamed to `predict.method` and `predict.prior`, respectively.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.qda")
lrn("classif.qda")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3learners**, **MASS**

Parameters

Id	Type	Default	Levels	Range
method	character	moment	moment, mle, mve, t	-
nu	integer	-		$(-\infty, \infty)$
predict.method	character	plug-in	plug-in, predictive, debiased	-
predict.prior	untyped	-		-
prior	untyped	-		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifQDA`

Methods

Public methods:

- `LearnerClassifQDA$new()`
- `LearnerClassifQDA$clone()`

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

Usage:

```
LearnerClassifQDA$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

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

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.qda")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_classif.ranger`
Ranger Classification Learner

Description

Random classification forest. Calls `ranger::ranger()` from package **ranger**.

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`:
  - Actual default: 2, using two threads, while also respecting environment variable `R_RANGER_NUM_THREADS`, `options(ranger.num.threads = N)`, or `options(Ncpus = N)`, with precedence in that order.
  - Adjusted value: 1.
  - Reason for change: Conflicting with parallelization via **future**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.ranger")
lrn("classif.ranger")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3learners**, **ranger**

Parameters

Id	Type	Default	Levels	Range
always.split.variables	untyped	-		-
class.weights	untyped	NULL		-
holdout	logical	FALSE	TRUE, FALSE	-

importance	character	-	none, impurity, impurity_corrected, permutation	-
keep.inbag	logical	FALSE	TRUE, FALSE	-
max.depth	integer	NULL		$[1, \infty)$
min.bucket	untyped	1L		-
min.node.size	untyped	NULL		-
mtry	integer	-		$[1, \infty)$
mtry.ratio	numeric	-		$[0, 1]$
na.action	character	na.learn	na.learn, na.omit, na.fail	-
num.random.splits	integer	1		$[1, \infty)$
node.stats	logical	FALSE	TRUE, FALSE	-
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, order, partition	-
sample.fraction	numeric	-		$[0, 1]$
save.memory	logical	FALSE	TRUE, FALSE	-
scale.permutation.importance	logical	FALSE	TRUE, FALSE	-
local.importance	logical	FALSE	TRUE, FALSE	-
seed	integer	NULL		$(-\infty, \infty)$
split.select.weights	untyped	NULL		-
splitrule	character	gini	gini, extratrees, hellinger	-
verbose	logical	TRUE	TRUE, FALSE	-
write.forest	logical	TRUE	TRUE, FALSE	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifRanger`

Methods

Public methods:

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

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

Usage:

```
LearnerClassifRanger$new()
```

`LearnerClassifRanger$importance()`: The importance scores are extracted from the model slot `variable.importance`. Parameter `importance.mode` must be set to `"impurity"`, `"impurity_corrected"`, or `"permutation"`

Usage:

```
LearnerClassifRanger$importance()
```

Returns: Named `numeric()`.

`LearnerClassifRanger$oob_error()`: The out-of-bag error, extracted from model slot `prediction.error`.

Usage:

```
LearnerClassifRanger$oob_error()
```

Returns: `numeric(1)`.

`LearnerClassifRanger$selected_features()`: The set of features used for node splitting in the forest.

Usage:

```
LearnerClassifRanger$selected_features()
```

Returns: `character()`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

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

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.

  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: mlr_learners_classif.cv_glmnet, mlr_learners_classif.glmnet, mlr_learners_classif.kknn, mlr_learners_classif.lda, mlr_learners_classif.log_reg, mlr_learners_classif.multinom, mlr_learners_classif.naive_bayes, mlr_learners_classif.nnet, mlr_learners_classif.qda, mlr_learners_classif.svm, mlr_learners_classif.xgboost, mlr_learners_regr.cv_glmnet, mlr_learners_regr.glmnet, mlr_learners_regr.kknn, mlr_learners_regr.km, mlr_learners_regr.lm, mlr_learners_regr.nnet, mlr_learners_regr.ranger, mlr_learners_regr.svm, mlr_learners_regr.xgboost

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.ranger")
learner$param_set$set_values(importance = "permutation")
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 the model
print(learner$model)

# Importance method
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.svm

Support Vector Machine

Description

Support vector machine for classification. Calls e1071::svm() from package **e1071**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.svm")
lrn("classif.svm")
```

Meta Information

- Task type: "classif"
- Predict Types: "response", "prob"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3learners**, **e1071**

Parameters

Id	Type	Default	Levels	Range
cachesize	numeric	40		$(-\infty, \infty)$
class.weights	untyped	NULL		-
coef0	numeric	0		$(-\infty, \infty)$
cost	numeric	1		$[0, \infty)$
cross	integer	0		$[0, \infty)$
decision.values	logical	FALSE	TRUE, FALSE	-
degree	integer	3		$[1, \infty)$
epsilon	numeric	0.1		$[0, \infty)$
fitted	logical	TRUE	TRUE, FALSE	-
gamma	numeric	-		$[0, \infty)$
kernel	character	radial	linear, polynomial, radial, sigmoid	-
nu	numeric	0.5		$(-\infty, \infty)$
scale	untyped	TRUE		-
shrinking	logical	TRUE	TRUE, FALSE	-
tolerance	numeric	0.001		$[0, \infty)$
type	character	C-classification	C-classification, nu-classification	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifSVM`

Methods

Public methods:

- `LearnerClassifSVM$new()`
- `LearnerClassifSVM$clone()`

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

Usage:

```
LearnerClassifSVM$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Cortes, Corinna, Vapnik, Vladimir (1995). "Support-vector networks." *Machine Learning*, **20**(3), 273–297. doi:10.1007/BF00994018.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: mlr_learners_classif.cv_glmnet, mlr_learners_classif.glmnet, mlr_learners_classif.kknn, mlr_learners_classif.lda, mlr_learners_classif.log_reg, mlr_learners_classif.multinom, mlr_learners_classif.naive_bayes, mlr_learners_classif.nnet, mlr_learners_classif.qda, mlr_learners_classif.ranger, mlr_learners_classif.xgboost, mlr_learners_regr.cv_glmnet, mlr_learners_regr.glmnet, mlr_learners_regr.kknn, mlr_learners_regr.km, mlr_learners_regr.lm, mlr_learners_regr.nnet, mlr_learners_regr.ranger, mlr_learners_regr.svm, mlr_learners_regr.xgboost

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.svm")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_classif.xgboost`
Extreme Gradient Boosting Classification Learner

Description

eXtreme Gradient Boosting classification. Calls `xgboost::xgb.train()` from package **xgboost**.

Note that using the `evals` parameter directly will lead to problems when wrapping this mlr3::Learner in a `mlr3pipelines` `GraphLearner` as the preprocessing steps will not be applied to the data in `evals`. See the section *Early Stopping and Validation* on how to do this.

Initial parameter values

- `nrounds`:
  - Actual default: no default.
  - Adjusted default: 1000.
  - Reason for change: Without a default construction of the learner would error. The lightgbm learner has a default of 1000, so we use the same here.
- `nthread`:
  - Actual value: Undefined, triggering auto-detection of the number of CPUs.
  - Adjusted value: 1.
  - Reason for change: Conflicting with parallelization via **future**.
- `verbose`:
  - Actual default: 1.
  - Adjusted default: 0.
  - Reason for change: Reduce verbosity.
- `verbosity`:

- Actual default: 1.
- Adjusted default: 0.
- Reason for change: Reduce verbosity.

Early Stopping and Validation

In order to monitor the validation performance during the training, you can set the `$validate` field of the Learner. For information on how to configure the validation set, see the *Validation* section of mlr3::Learner. This validation data can also be used for early stopping, which can be enabled by setting the `early_stopping_rounds` parameter. The final (or in the case of early stopping best) validation scores can be accessed via `$internal_valid_scores`, and the optimal `nrounds` via `$internal_tuned_values`. The internal validation measure can be set via the `custom_metric` parameter that can be a mlr3::Measure, a function, or a character string for the internal xgboost measures. Using an mlr3::Measure is slower than the internal xgboost measures, but allows to use the same measure for tuning and validation.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("classif.xgboost")
lrn("classif.xgboost")
```

Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3learners**, **xgboost**

Parameters

Id	Type	Default	Levels	Range
alpha	numeric	0		$[0, \infty)$
approxcontrib	logical	FALSE	TRUE, FALSE	-
base_score	numeric	-		$(-\infty, \infty)$
booster	character	gbtree	gbtree, gblinear, dart	-
callbacks	untyped	list()		-
colsample_bylevel	numeric	1		$[0, 1]$
colsample_bynode	numeric	1		$[0, 1]$
colsample_bytree	numeric	1		$[0, 1]$
device	untyped	"cpu"		-
disable_default_eval_metric	logical	FALSE	TRUE, FALSE	-
early_stopping_rounds	integer	NULL		$[1, \infty)$
eta	numeric	0.3		$[0, 1]$
evals	untyped	NULL		-

eval_metric	untyped	-		-
custom_metric	untyped	-		-
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_cached_hist_node	integer	65536		$(-\infty, \infty)$
max_cat_to_onehot	integer	-		$(-\infty, \infty)$
max_cat_threshold	numeric	-		$(-\infty, \infty)$
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		$[0, \infty)$
missing	numeric	NA		$(-\infty, \infty)$
monotone_constraints	untyped	0		-
nrounds	integer	-		$[1, \infty)$
normalize_type	character	tree	tree, forest	-
nthread	integer	-		$[1, \infty)$
num_parallel_tree	integer	1		$[1, \infty)$
objective	untyped	"binary:logistic"		-
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	-
seed	integer	-		$(-\infty, \infty)$
seed_per_iteration	logical	FALSE	TRUE, FALSE	-
sampling_method	character	uniform	uniform, gradient_based	-
sample_type	character	uniform	uniform, weighted	-
save_name	untyped	NULL		-
save_period	integer	NULL		$[0, \infty)$
scale_pos_weight	numeric	1		$(-\infty, \infty)$
skip_drop	numeric	0		$[0, 1]$
subsample	numeric	1		$[0, 1]$
top_k	integer	0		$[0, \infty)$
training	logical	FALSE	TRUE, FALSE	-
tree_method	character	auto	auto, exact, approx, hist, gpu_hist	-
tweedie_variance_power	numeric	1.5		$[1, 2]$
updater	untyped	-		-
use_rmm	logical	-	TRUE, FALSE	-
validate_features	logical	TRUE	TRUE, FALSE	-
verbose	integer	-		$[0, 2]$
verbosity	integer	-		$[0, 2]$
xgb_model	untyped	NULL		-
use_pred_offset	logical	-	TRUE, FALSE	-

Offset

If a Task has a column with the role offset, it will automatically be used during training. The offset is incorporated through the xgboost::xgb.DMatrix interface, using the base_margin field. During prediction, the offset column from the test set is used only if use_pred_offset = TRUE (default) and the Task has a column with the role offset. The test set offsets are passed via the base_margin argument in xgboost::predict.xgb.Booster(). Otherwise, if the user sets use_pred_offset = FALSE (or the Task doesn't have a column with the offset role), the (possibly estimated) global intercept from the train set is applied. See https://xgboost.readthedocs.io/en/stable/tutorials/intercept.html.

Super classes

mlr3::Learner -> mlr3::LearnerClassif -> LearnerClassifXgboost

Active bindings

internal_valid_scores (named list() or NULL) The validation scores extracted from model$evaluation_log. If early stopping is activated, this contains the validation scores of the model for the optimal nrounds, otherwise the scores are taken from the final boosting round nrounds.

best_valid_scores (named list() or NULL) The validation scores of the best boosting round, extracted from model$evaluation_log. Because XGBoost predicts with the best nrounds by default, 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, in which case msr("best_valid_score") cannot be computed.

internal_tuned_values (named list() or NULL) If early stopping is activated, this returns a list with nrounds, which is extracted from $best_iteration of the model and otherwise NULL.

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".

model (any)
The fitted model. Only available after $train() has been called.

Methods

Public methods:

- LearnerClassifXgboost$new()
- LearnerClassifXgboost$importance()
- LearnerClassifXgboost$clone()

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

Usage:

```
LearnerClassifXgboost$new()
```

`LearnerClassifXgboost$importance()`: The importance scores are calculated with `xgboost::xgb.importance()`.

Usage:

```
LearnerClassifXgboost$importance()
```

Returns: Named `numeric()`.

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

Usage:

```
LearnerClassifXgboost$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.

The `outputmargin`, `predcontrib`, `predinteraction`, and `predleaf` parameters are not supported. You can still call e.g. `predict(learner$model, newdata = newdata, outputmargin = TRUE)` to get these predictions.

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

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("classif.xgboost")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

# Score the predictions
predictions$score()

# Early stopping
learner = lrn("classif.xgboost", nrounds = 100, early_stopping_rounds = 10, validate = 0.3)

# Train learner with early stopping
learner$train(task)

# Inspect optimal nrounds and validation performance
learner$internal_tuned_values
learner$internal_valid_scores
```

---

mlr_learners_regr.cv_glmnet

GLM with Elastic Net Regularization Regression Learner

Description

Generalized linear models with elastic net regularization. Calls `glmnet::cv.glmnet()` from package **glmnet**.

Supported `family` values are `"gaussian"` and `"poisson"`. The default for the hyperparameter `family` is `"gaussian"`.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("regr.cv_glmnet")
lrn("regr.cv_glmnet")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3learners**, **glmnet**

Parameters

Id	Type	Default	Levels	Range
lambda	untyped	NULL		-
type.measure	character	deviance	deviance, mse, mae	-
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		$[0, 1]$
family	character	-	gaussian, poisson	-
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		-
type.gaussian	character	-	covariance, naive	-
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		$(-\infty, \infty)$
pmax	integer	NULL		$(-\infty, \infty)$
s	numeric	lambda.1se		$[0, \infty)$
predict.gamma	numeric	gamma.1se		$[0, 1]$
exact	logical	FALSE	TRUE, FALSE	-
use_pred_offset	logical	-	TRUE, FALSE	-
seed	integer	-		$(-\infty, \infty)$

Custom mlr3 parameters

- `seed`:
  - Optional integer used to seed the call to `glmnet::cv.glmnet()`, making its random fold assignment, and therefore the selected lambda, reproducible.
  - The global random state is reset afterwards, so it is left unchanged.
  - Defaults to `NA`, in which case no seed is set and the global random state is used.

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.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrCVGlmnet`

Methods

Public methods:

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

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

Usage:

```
LearnerRegrCVGlmnet$new()
```

`LearnerRegrCVGlmnet$selected_features()`: Returns the set of selected features as reported by `glmnet::predict.glmnet()` with `type` set to `"nonzero"`.

Usage:

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

Arguments:

`lambda` (`numeric(1)`)
Custom `lambda`, defaults to the active lambda depending on parameter set.

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

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

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("regr.cv_glmnet")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_regr.glmnet`

GLM with Elastic Net Regularization Regression Learner

Description

Generalized linear models with elastic net regularization. Calls `glmnet::glmnet()` from package **glmnet**.

Supported `family` values are `"gaussian"` and `"poisson"`. The default for the hyperparameter `family` is `"gaussian"`.

Details

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 an 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 counterparts in `mlr_learners_classif.cv_glmnet()` or `mlr_learners_regr.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).

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("regr.glmnet")
lrn("regr.glmnet")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3learners**, **glmnet**

Parameters

Id	Type	Default	Levels	Range
family	character	-	gaussian, poisson	-
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		-
type.gaussian	character	-	covariance, naive	-
relax	logical	FALSE	TRUE, FALSE	-
trace.it	integer	0		$[0, 1]$
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		$[1, \infty)$
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		$[0, 1]$
use_pred_offset	logical	-	TRUE, FALSE	-

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.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrGlmnet`

Methods

Public methods:

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

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

Usage:

```
LearnerRegrGlmnet$new()
```

`LearnerRegrGlmnet$selected_features()`**:** Returns the set of selected features as reported by `glmnet::predict.glmnet()` with `type` set to `"nonzero"`.

Usage:

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

Arguments:

`lambda (numeric(1))`
Custom `lambda`, defaults to the active lambda depending on parameter set.

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

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

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

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("regr.glmnet")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_regr.kknn`

k-Nearest-Neighbor Regression Learner

Description

k-Nearest-Neighbor regression. Calls `kknn::kknn()` from package **kknn**.

Initial parameter values

- `store_model`:
  - See note.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("regr.kknn")
lrn("regr.kknn")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3learners**, **kknn**

Parameters

Id	Type	Default	Levels
k	integer	7	
distance	numeric	2	
kernel	character	optimal	rectangular, triangular, epanechnikov, biweight, triweight, cos, inv, gaussian, rank, optim
scale	logical	TRUE	TRUE, FALSE
ykernel	untyped	NULL	
store_model	logical	FALSE	TRUE, FALSE

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrKKNN`

Methods

Public methods:

- `LearnerRegrKKNN$new()`
- `LearnerRegrKKNN$clone()`

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

Usage:

```
LearnerRegrKKNN$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

There is no training step for k-NN models, just storing the training data to process it during the predict step. Therefore, `$model` returns a list with the following elements:

- `formula`: Formula for calling `kknn::kknn()` during `$predict()`.
- `data`: Training data for calling `kknn::kknn()` during `$predict()`.
- `pv`: Training parameters for calling `kknn::kknn()` during `$predict()`.
- `kknn`: Model as returned by `kknn::kknn()`, only available **after** `$predict()` has been called. This is not stored by default, you must set hyperparameter `store_model` to `TRUE`.

References

Hechenbichler, Klaus, Schliep, Klaus (2004). "Weighted k-nearest-neighbor techniques and ordinal classification." Technical Report Discussion Paper 399, SFB 386, Ludwig-Maximilians University Munich. doi:10.5282/ubm/epub.1769.

Samworth, J R (2012). "Optimal weighted nearest neighbour classifiers." *The Annals of Statistics*, **40**(5), 2733–2763. doi:10.1214/12AOS1049.

Cover, Thomas, Hart, Peter (1967). "Nearest neighbor pattern classification." *IEEE transactions on information theory*, **13**(1), 21–27. doi:10.1109/TIT.1967.1053964.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: mlr_learners_classif.cv_glmnet, mlr_learners_classif.glmnet, mlr_learners_classif.kknn, mlr_learners_classif.lda, mlr_learners_classif.log_reg, mlr_learners_classif.multinom, mlr_learners_classif.naive_bayes, mlr_learners_classif.nnet, mlr_learners_classif.qda, mlr_learners_classif.ranger, mlr_learners_classif.svm, mlr_learners_classif.xgboost, mlr_learners_regr.cv_glmnet, mlr_learners_regr.glmnet, mlr_learners_regr.km, mlr_learners_regr.lm, mlr_learners_regr.nnet, mlr_learners_regr.ranger, mlr_learners_regr.svm, mlr_learners_regr.xgboost

Examples

```
# Define the Learner and set parameter values
learner = lrn("regr.kknn")
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 the model
```

```
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_regr.km` *Kriging Regression Learner*

Description

Kriging regression. Calls `DiceKriging::km()` from package **DiceKriging**.

- The predict type hyperparameter "type" defaults to "SK" (simple kriging).
- The additional hyperparameter `nugget.stability` is used to overwrite the hyperparameter `nugget` with `nugget.stability * var(y)` before training to improve the numerical stability. We recommend a value of `1e-8`.
- The additional hyperparameter `jitter` can be set to add `N(0, [jitter])`-distributed noise to the data before prediction to avoid perfect interpolation. We recommend a value of `1e-12`.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("regr.km")
lrn("regr.km")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “se”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: **mlr3**, **mlr3learners**, **DiceKriging**

Parameters

Id	Type	Default	Levels	Range
bias.correct	logical	FALSE	TRUE, FALSE	-
checkNames	logical	TRUE	TRUE, FALSE	-
coef.cov	untyped	NULL		-
coef.trend	untyped	NULL		-
coef.var	untyped	NULL		-
control	untyped	NULL		-
cov.compute	logical	TRUE	TRUE, FALSE	-
covtype	character	matern5_2	gauss, matern5_2, matern3_2, exp, powexp	-
estim.method	character	MLE	MLE, LOO	-
gr	logical	TRUE	TRUE, FALSE	-
iso	logical	FALSE	TRUE, FALSE	-
jitter	numeric	0		$[0, \infty)$
kernel	untyped	NULL		-
knots	untyped	NULL		-
light.return	logical	FALSE	TRUE, FALSE	-
lower	untyped	NULL		-
multistart	integer	1		$(-\infty, \infty)$
noise.var	untyped	NULL		-
nugget	numeric	-		$(-\infty, \infty)$
nugget.estim	logical	FALSE	TRUE, FALSE	-
nugget.stability	numeric	0		$[0, \infty)$
optim.method	character	BFGS	BFGS, gen	-
parinit	untyped	NULL		-
penalty	untyped	NULL		-
scaling	logical	FALSE	TRUE, FALSE	-
se.compute	logical	TRUE	TRUE, FALSE	-
type	character	SK	SK, UK	-
upper	untyped	NULL		-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrKM`

Methods

Public methods:

- `LearnerRegrKM$new()`
- `LearnerRegrKM$clone()`

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

Usage:

```
LearnerRegrKM$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Roustant O, Ginsbourger D, Deville Y (2012). "DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization." *Journal of Statistical Software*, **51**(1), 1–55. doi:10.18637/jss.v051.i01.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("regr.km")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_regr.lm` *Linear Model Regression Learner*

Description

Ordinary linear regression. Calls `stats::lm()`.

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::lm()`. 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.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("regr.lm")
lrn("regr.lm")
```

Meta Information

- Task type: “regr”
- Predict Types: “response”, “se”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”
- Required Packages: **mlr3**, **mlr3learners**, ’stats’

Parameters

Id	Type	Default	Levels	Range
df	numeric	Inf		$(-\infty, \infty)$
interval	character	-	none, confidence, prediction	-
level	numeric	0.95		$(-\infty, \infty)$
model	logical	TRUE	TRUE, FALSE	-
pred.var	untyped	-		-
qr	logical	TRUE	TRUE, FALSE	-
scale	numeric	NULL		$(-\infty, \infty)$
singular.ok	logical	TRUE	TRUE, FALSE	-
x	logical	FALSE	TRUE, FALSE	-
y	logical	FALSE	TRUE, FALSE	-
rankdeficient	character	-	warnif, simple, non-estim, NA, NAwarn	-
tol	numeric	1e-07		$(-\infty, \infty)$
verbose	logical	FALSE	TRUE, FALSE	-
use_pred_offset	logical	TRUE	TRUE, FALSE	-

Contrasts

To ensure reproducibility, this learner always uses the default contrasts:

- `contr.treatment()` for unordered factors, and
- `contr.poly()` for ordered factors.

Setting the option `"contrasts"` does not have any effect. Instead, set the respective hyperparameter or use **mlr3pipelines** to create dummy features.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrLM`

Methods

Public methods:

- `LearnerRegrLM$new()`
- `LearnerRegrLM$clone()`

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

Usage:

```
LearnerRegrLM$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.svm`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("regr.lm")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_regr.nnet`

Neural Network Regression Learner

Description

Single Layer Neural Network. Calls `nnet::nnet.formula()` from package **nnet**.

Note that modern neural networks with multiple layers are connected via package mlr3torch.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("regr.nnet")
lrn("regr.nnet")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3learners**, **nnet**

Parameters

Id	Type	Default	Levels	Range
Hess	logical	FALSE	TRUE, FALSE	-
MaxNWts	integer	1000		$[1, \infty)$
Wts	untyped	-		-
abstol	numeric	1e-04		$(-\infty, \infty)$
censored	logical	FALSE	TRUE, FALSE	-
contrasts	untyped	NULL		-
decay	numeric	0		$(-\infty, \infty)$
mask	untyped	-		-
maxit	integer	100		$[1, \infty)$
na.action	untyped	-		-
rang	numeric	0.7		$(-\infty, \infty)$
reltol	numeric	1e-08		$(-\infty, \infty)$
size	integer	3		$[0, \infty)$
skip	logical	FALSE	TRUE, FALSE	-
subset	untyped	-		-
trace	logical	TRUE	TRUE, FALSE	-
formula	untyped	-		-

Initial parameter values

- `size`:
  - Adjusted default: 3L.
  - Reason for change: no default in `nnet()`.

Custom mlr3 parameters

- `formula`: if not provided, the formula is set to `task$formula()`.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrNnet`

Methods

Public methods:

- `LearnerRegrNnet$new()`
- `LearnerRegrNnet$clone()`

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

Usage:

```
LearnerRegrNnet$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Ripley BD (1996). *Pattern Recognition and Neural Networks*. Cambridge University Press. doi:10.1017/cbo9780511812651.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.

  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: mlr_learners_classif.cv_glmnet, mlr_learners_classif.glmnet, mlr_learners_classif.kknn, mlr_learners_classif.lda, mlr_learners_classif.log_reg, mlr_learners_classif.multinom, mlr_learners_classif.naive_bayes, mlr_learners_classif.nnet, mlr_learners_classif.qda, mlr_learners_classif.ranger, mlr_learners_classif.svm, mlr_learners_classif.xgboost, mlr_learners_regr.cv_glmnet, mlr_learners_regr.glmnet, mlr_learners_regr.kknn, mlr_learners_regr.km, mlr_learners_regr.lm, mlr_learners_regr.ranger, mlr_learners_regr.svm, mlr_learners_regr.xgboost

Examples

```
# Define the Learner and set parameter values
learner = lrn("regr.nnet")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

---

mlr_learners_regr.ranger

Ranger Regression Learner

Description

Random regression forest. Calls `ranger()` from package **ranger**.

Details

In addition to the uncertainty estimation methods provided by the ranger package, the learner provides an ensemble standard deviation and law of total variance uncertainty estimation. Both methods compute the empirical mean and variance of the training data points that fall into the predicted leaf nodes. The ensemble standard deviation method calculates the standard deviation of the mean of the leaf nodes. The law of total variance method calculates the mean of the variance of the leaf nodes plus the variance of the means of the leaf nodes. Formulas for the ensemble standard deviation and law of total variance method are given in Hutter et al. (2015).

For these 2 methods, the parameter `sigma2.threshold` can be used to set a threshold for the variance of the leaf nodes, this is a minimal value for the variance of the leaf nodes, if the variance is below this threshold, it is set to this value (as described in the paper). Default is 1e-2.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("regr.ranger")
lrn("regr.ranger")
```

Meta Information

- Task type: "regr"
- Predict Types: "response", "se", "quantiles"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered"
- Required Packages: **mlr3**, **mlr3learners**, **ranger**

Parameters

Id	Type	Default	Levels
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	
min.bucket	integer	1	
min.node.size	integer	5	
mtry	integer	-	
mtry.ratio	numeric	-	
na.action	character	na.learn	na.learn, na.omit, na.fail
node.stats	logical	FALSE	TRUE, FALSE
num.random.splits	integer	1	
num.threads	integer	1	
num.trees	integer	500	
oob.error	logical	TRUE	TRUE, FALSE
poisson.tau	numeric	1	
regularization.factor	untyped	1	

regularization.usedepth	logical	FALSE	TRUE, FALSE
replace	logical	TRUE	TRUE, FALSE
respect.unordered.factors	character	-	ignore, order, partition
sample.fraction	numeric	-	
save.memory	logical	FALSE	TRUE, FALSE
scale.permutation.importance	logical	FALSE	TRUE, FALSE
local.importance	logical	FALSE	TRUE, FALSE
se.method	character	infjack	jack, infjack, ensemble_standard_deviation, law_of_total_variance
sigma2.threshold	numeric	0.01	
seed	integer	NULL	
split.select.weights	untyped	NULL	
splitrule	character	variance	variance, extratrees, maxstat, beta, poisson
verbose	logical	TRUE	TRUE, FALSE
write.forest	logical	TRUE	TRUE, FALSE

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`:
  - Actual default: 2, using two threads, while also respecting environment variable `R_RANGER_NUM_THREADS`, `options(ranger.num.threads = N)`, or `options(Ncpus = N)`, with precedence in that order.
  - Adjusted value: 1.
  - Reason for change: Conflicting with parallelization via **future**.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrRanger`

Active bindings

`native_model` (ranger::ranger)
The fitted model.

Methods

Public methods:

- `LearnerRegrRanger$new()`
- `LearnerRegrRanger$importance()`

- LearnerRegrRanger$oob_error()
- LearnerRegrRanger$selected_features()
- LearnerRegrRanger$clone()

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

Usage:

```
LearnerRegrRanger$new()
```

`LearnerRegrRanger$importance()`: The importance scores are extracted from the model slot `variable.importance`. Parameter `importance.mode` must be set to `"impurity"`, `"impurity_corrected"`, or `"permutation"`

Usage:

```
LearnerRegrRanger$importance()
```

Returns: Named `numeric()`.

`LearnerRegrRanger$oob_error()`: The out-of-bag error, extracted from model slot `prediction.error`.

Usage:

```
LearnerRegrRanger$oob_error()
```

Returns: `numeric(1)`

`LearnerRegrRanger$selected_features()`: The set of features used for node splitting in the forest.

Usage:

```
LearnerRegrRanger$selected_features()
```

Returns: `character()`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

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.

Hutter, Frank, Xu, Lin, Hoos, H. H, Leyton-Brown, Kevin (2015). "Algorithm runtime prediction: methods and evaluation." In *Proceedings of the 24th International Conference on Artificial Intelligence*, series IJCAI'15, 4197–4201.

See Also

- Chapter in the mlr3book: https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: mlr_learners_classif.cv_glmnet, mlr_learners_classif.glmnet, mlr_learners_classif.kknn, mlr_learners_classif.lda, mlr_learners_classif.log_reg, mlr_learners_classif.multinom, mlr_learners_classif.naive_bayes, mlr_learners_classif.nnet, mlr_learners_classif.qda, mlr_learners_classif.ranger, mlr_learners_classif.svm, mlr_learners_classif.xgboost, mlr_learners_regr.cv_glmnet, mlr_learners_regr.glmnet, mlr_learners_regr.kknn, mlr_learners_regr.km, mlr_learners_regr.lm, mlr_learners_regr.nnet, mlr_learners_regr.svm, mlr_learners_regr.xgboost

Examples

```
# Define the Learner and set parameter values
learner = lrn("regr.ranger")
learner$param_set$set_values(importance = "permutation")
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 the model
print(learner$model)

# Importance method
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.svm` *Support Vector Machine*

Description

Support vector machine for regression. Calls `e1071::svm()` from package **e1071**.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("regr.svm")
lrn("regr.svm")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3learners**, **e1071**

Parameters

Id	Type	Default	Levels	Range
cachesize	numeric	40		$(-\infty, \infty)$
coef0	numeric	0		$(-\infty, \infty)$
cost	numeric	1		$[0, \infty)$
cross	integer	0		$[0, \infty)$
degree	integer	3		$[1, \infty)$
epsilon	numeric	0.1		$[0, \infty)$
fitted	logical	TRUE	TRUE, FALSE	-
gamma	numeric	-		$[0, \infty)$
kernel	character	radial	linear, polynomial, radial, sigmoid	-
nu	numeric	0.5		$(-\infty, \infty)$
scale	untyped	TRUE		-
shrinking	logical	TRUE	TRUE, FALSE	-
tolerance	numeric	0.001		$[0, \infty)$
type	character	eps-regression	eps-regression, nu-regression	-

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrSVM`

Methods

Public methods:

- `LearnerRegrSVM$new()`
- `LearnerRegrSVM$clone()`

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

Usage:

```
LearnerRegrSVM$new()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

Cortes, Corinna, Vapnik, Vladimir (1995). "Support-vector networks." *Machine Learning*, **20**(3), 273–297. doi:10.1007/BF00994018.

See Also

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: `mlr_learners_classif.cv_glmnet`, `mlr_learners_classif.glmnet`, `mlr_learners_classif.kknn`, `mlr_learners_classif.lda`, `mlr_learners_classif.log_reg`, `mlr_learners_classif.multinom`, `mlr_learners_classif.naive_bayes`, `mlr_learners_classif.nnet`, `mlr_learners_classif.qda`, `mlr_learners_classif.ranger`, `mlr_learners_classif.svm`, `mlr_learners_classif.xgboost`, `mlr_learners_regr.cv_glmnet`, `mlr_learners_regr.glmnet`, `mlr_learners_regr.kknn`, `mlr_learners_regr.km`, `mlr_learners_regr.lm`, `mlr_learners_regr.nnet`, `mlr_learners_regr.ranger`, `mlr_learners_regr.xgboost`

Examples

```
# Define the Learner and set parameter values
learner = lrn("regr.svm")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

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

`mlr_learners_regr.xgboost`

Extreme Gradient Boosting Regression Learner

Description

eXtreme Gradient Boosting regression. Calls `xgboost::xgb.train()` from package **xgboost**.

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`.

Note that using the `evals` parameter directly will lead to problems when wrapping this mlr3::Learner in a `mlr3pipelines` `GraphLearner` as the preprocessing steps will not be applied to the data in `evals`. See the section *Early Stopping and Validation* on how to do this.

Offset

If a `Task` has a column with the role `offset`, it will automatically be used during training. The offset is incorporated through the xgboost::xgb.DMatrix interface, using the `base_margin` field. During prediction, the offset column from the test set is used only if `use_pred_offset` = `TRUE` (default) and the `Task` has a column with the role `offset`. The test set offsets are passed via the `base_margin` argument in `xgboost::predict.xgb.Booster()`. Otherwise, if the user sets `use_pred_offset` = `FALSE` (or the `Task` doesn't have a column with the `offset` role), the (possibly

estimated) global intercept from the train set is applied. See https://xgboost.readthedocs.io/en/stable/tutorials/intercept.html.

Dictionary

This mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function `mlr3::lrn()`:

```
mlr_learners$get("regr.xgboost")
lrn("regr.xgboost")
```

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: **mlr3**, **mlr3learners**, **xgboost**

Parameters

Id	Type	Default	Levels	Range
alpha	numeric	0		$[0, \infty)$
approxcontrib	logical	FALSE	TRUE, FALSE	-
base_score	numeric	-		$(-\infty, \infty)$
booster	character	gbtree	gbtree, gblinear, dart	-
callbacks	untyped	list()		-
colsample_bylevel	numeric	1		$[0, 1]$
colsample_bynode	numeric	1		$[0, 1]$
colsample_bytree	numeric	1		$[0, 1]$
device	untyped	"cpu"		-
disable_default_eval_metric	logical	FALSE	TRUE, FALSE	-
early_stopping_rounds	integer	NULL		$[1, \infty)$
eta	numeric	0.3		$[0, 1]$
evals	untyped	NULL		-
eval_metric	untyped	-		-
custom_metric	untyped	-		-
feature_selector	character	cyclic	cyclic, shuffle, random, greedy, thrifty	-
gamma	numeric	0		$[0, \infty)$
grow_policy	character	depthwise	depthwise, lossguide	-
huber_slope	numeric	1		$(-\infty, \infty)$
interaction_constraints	untyped	-		-
iterationrange	untyped	-		-
lambda	numeric	1		$[0, \infty)$
max_bin	integer	256		$[2, \infty)$
max_cached_hist_node	integer	65536		$(-\infty, \infty)$
max_cat_to_onehot	integer	-		$(-\infty, \infty)$
max_cat_threshold	numeric	-		$(-\infty, \infty)$

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		$[0, \infty)$
missing	numeric	NA		$(-\infty, \infty)$
monotone_constraints	untyped	0		-
nrounds	integer	-		$[1, \infty)$
normalize_type	character	tree	tree, forest	-
nthread	integer	-		$[1, \infty)$
num_parallel_tree	integer	1		$[1, \infty)$
objective	untyped	"reg:squarederror"		-
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	-
seed	integer	-		$(-\infty, \infty)$
seed_per_iteration	logical	FALSE	TRUE, FALSE	-
sampling_method	character	uniform	uniform, gradient_based	-
sample_type	character	uniform	uniform, weighted	-
save_name	untyped	NULL		-
save_period	integer	NULL		$[0, \infty)$
scale_pos_weight	numeric	1		$(-\infty, \infty)$
skip_drop	numeric	0		$[0, 1]$
subsample	numeric	1		$[0, 1]$
top_k	integer	0		$[0, \infty)$
training	logical	FALSE	TRUE, FALSE	-
tree_method	character	auto	auto, exact, approx, hist, gpu_hist	-
tweedie_variance_power	numeric	1.5		$[1, 2]$
updater	untyped	-		-
use_rmm	logical	-	TRUE, FALSE	-
validate_features	logical	TRUE	TRUE, FALSE	-
verbose	integer	-		$[0, 2]$
verbosity	integer	-		$[0, 2]$
xgb_model	untyped	NULL		-
use_pred_offset	logical	-	TRUE, FALSE	-

Early Stopping and Validation

In order to monitor the validation performance during the training, you can set the `$validate` field of the Learner. For information on how to configure the validation set, see the *Validation* section of mlr3::Learner. This validation data can also be used for early stopping, which can be enabled by setting the `early_stopping_rounds` parameter. The final (or in the case of early stopping best) validation scores can be accessed via `$internal_valid_scores`, and the optimal `nrounds` via `$internal_tuned_values`. The internal validation measure can be set via the `custom_metric`

parameter that can be a mlr3::Measure, a function, or a character string for the internal xgboost measures. Using an mlr3::Measure is slower than the internal xgboost measures, but allows to use the same measure for tuning and validation.

Initial parameter values

- `nrounds`:
  - Actual default: no default.
  - Adjusted default: 1000.
  - Reason for change: Without a default construction of the learner would error. The lightgbm learner has a default of 1000, so we use the same here.
- `nthread`:
  - Actual value: Undefined, triggering auto-detection of the number of CPUs.
  - Adjusted value: 1.
  - Reason for change: Conflicting with parallelization via **future**.
- `verbose`:
  - Actual default: 1.
  - Adjusted default: 0.
  - Reason for change: Reduce verbosity.
- `verbosity`:
  - Actual default: 1.
  - Adjusted default: 0.
  - Reason for change: Reduce verbosity.

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> `LearnerRegrXgboost`

Active bindings

`internal_valid_scores` (named `list()` or `NULL`) The validation scores extracted from `model$evaluation_log`. If early stopping is activated, this contains the validation scores of the model for the optimal `nrounds`, otherwise the `nrounds` for the final model.

`best_valid_scores` (named `list()` or `NULL`) The validation scores of the best boosting round, extracted from `model$evaluation_log`. Because XGBoost predicts with the best `nrounds` by default, these are identical to `$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, in which case `msr("best_valid_score")` cannot be computed.

`internal_tuned_values` (named `list()` or `NULL`) If early stopping is activated, this returns a list with `nrounds`, which is extracted from `$best_iteration` of the model and otherwise `NULL`.

`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"`.

`model` (any)
The fitted model. Only available after `$train()` has been called. Returns the `$best_iteration` when early stopping is activated.

Methods

Public methods:

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

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

Usage:

```
LearnerRegrXgboost$new()
```

`LearnerRegrXgboost$importance()`: The importance scores are calculated with `xgboost::xgb.importance()`.

Usage:

```
LearnerRegrXgboost$importance()
```

Returns: Named `numeric()`.

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

Usage:

```
LearnerRegrXgboost$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`.

The `outputmargin`, `predcontrib`, `predinteraction`, and `predleaf` parameters are not supported. You can still call e.g. `predict(learner$model, newdata = newdata, outputmargin = TRUE)` to get these predictions.

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

- Chapter in the mlr3book: `https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners`
- Package mlr3extralearners for more learners.
- 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).
- **mlr3pipelines** to combine learners with pre- and postprocessing steps.

- Extension packages for additional task types:
  - **mlr3proba** for probabilistic supervised regression and survival analysis.
  - **mlr3cluster** for unsupervised clustering.
- **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.

Other Learner: mlr_learners_classif.cv_glmnet, mlr_learners_classif.glmnet, mlr_learners_classif.kknn, mlr_learners_classif.lda, mlr_learners_classif.log_reg, mlr_learners_classif.multinom, mlr_learners_classif.naive_bayes, mlr_learners_classif.nnet, mlr_learners_classif.qda, mlr_learners_classif.ranger, mlr_learners_classif.svm, mlr_learners_classif.xgboost, mlr_learners_regr.cv_glmnet, mlr_learners_regr.glmnet, mlr_learners_regr.kknn, mlr_learners_regr.km, mlr_learners_regr.lm, mlr_learners_regr.nnet, mlr_learners_regr.ranger, mlr_learners_regr.svm

Examples

```
# Define the Learner and set parameter values
learner = lrn("regr.xgboost")
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 the model
print(learner$model)

# Importance method
if ("importance" %in% learner$properties) print(learner$importance())

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

# Score the predictions
predictions$score()

# Early stopping
learner = lrn("regr.xgboost", nrounds = 100, early_stopping_rounds = 10, validate = 0.3)

# Train learner with early stopping
learner$train(task)

# Inspect optimal nrounds and validation performance
learner$internal_tuned_values
learner$internal_valid_scores
```

Index

∗ **Learner**
mlr_learners_classif.cv_glmnet, 4
mlr_learners_classif.glmnet, 7
mlr_learners_classif.kknn, 11
mlr_learners_classif.lda, 14
mlr_learners_classif.log_reg, 17
mlr_learners_classif.multinom, 20
mlr_learners_classif.naive_bayes, 22
mlr_learners_classif.nnet, 24
mlr_learners_classif.qda, 27
mlr_learners_classif.ranger, 30
mlr_learners_classif.svm, 33
mlr_learners_classif.xgboost, 36
mlr_learners_regr.cv_glmnet, 41
mlr_learners_regr.glmnet, 45
mlr_learners_regr.kknn, 49
mlr_learners_regr.km, 52
mlr_learners_regr.lm, 55
mlr_learners_regr.nnet, 58
mlr_learners_regr.ranger, 60
mlr_learners_regr.svm, 65
mlr_learners_regr.xgboost, 67

contr.poly(), *18*, *56*
contr.treatment(), *18*, *56*

DiceKriging::km(), *52*
Dictionary, *6*, *10*, *13*, *16*, *19*, *21*, *23*, *26*, *29*, *32*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *59*, *64*, *66*, *71*
dictionary, *4*, *8*, *12*, *14*, *17*, *20*, *22*, *25*, *27*, *30*, *34*, *37*, *42*, *46*, *49*, *52*, *55*, *58*, *61*, *65*, *68*

e1071::naiveBayes(), *22*
e1071::svm(), *33*, *65*

glmnet::cv.glmnet(), *4*, *8*, *41*, *43*, *45*
glmnet::glmnet(), *4*, *8*, *9*, *43*, *45*, *47*
glmnet::predict.glmnet(), *4*, *6*, *9*, *10*, *43*, *44*, *47*

kknn::kknn(), *11*, *13*, *49*, *50*

LearnerClassifCVGlmnet *(*mlr_learners_classif.cv_glmnet*)*, 4
LearnerClassifGlmnet *(*mlr_learners_classif.glmnet*)*, 7
LearnerClassifKKNN *(*mlr_learners_classif.kknn*)*, 11
LearnerClassifLDA *(*mlr_learners_classif.lda*)*, 14
LearnerClassifLogReg *(*mlr_learners_classif.log_reg*)*, 17
LearnerClassifMultinom *(*mlr_learners_classif.multinom*)*, 20
LearnerClassifNaiveBayes *(*mlr_learners_classif.naive_bayes*)*, 22
LearnerClassifNnet *(*mlr_learners_classif.nnet*)*, 24
LearnerClassifQDA *(*mlr_learners_classif.qda*)*, 27
LearnerClassifRanger *(*mlr_learners_classif.ranger*)*, 30
LearnerClassifSVM *(*mlr_learners_classif.svm*)*, 33
LearnerClassifXgboost *(*mlr_learners_classif.xgboost*)*, 36
LearnerRegrCVGlmnet *(*mlr_learners_regr.cv_glmnet*)*, 41

LearnerRegrGlmnet
(mlr_learners_regr.glmnet), 45
LearnerRegrKKNN
(mlr_learners_regr.kknn), 49
LearnerRegrKM (mlr_learners_regr.km), 52
LearnerRegrLM (mlr_learners_regr.lm), 55
LearnerRegrNnet
(mlr_learners_regr.nnet), 58
LearnerRegrRanger
(mlr_learners_regr.ranger), 60
LearnerRegrSVM (mlr_learners_regr.svm), 65
LearnerRegrXgboost
(mlr_learners_regr.xgboost), 67
Learners, *6, 7*, *10*, *13*, *16*, *19*, *21*, *23*, *26*, *29*, *32*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *59*, *64*, *66*, *71*

MASS::lda(), *14*
MASS::qda(), *27*
mlr3::Learner, *4*, *6*, *8, 9*, *12*, *14, 15*, *17, 18*, *20–23*, *25–28*, *30, 31*, *34*, *36, 37*, *39*, *42, 43*, *46, 47*, *49, 50*, *52, 53*, *55, 56*, *58, 59*, *61, 62*, *65*, *67–70*
mlr3::LearnerClassif, *6*, *9*, *12*, *15*, *18*, *21*, *23*, *26*, *28*, *31*, *34*, *39*
mlr3::LearnerRegr, *43*, *47*, *50*, *53*, *56*, *59*, *62*, *65*, *70*
mlr3::lrn(), *4*, *8*, *12*, *14*, *17*, *20*, *22*, *25*, *27*, *30*, *34*, *37*, *42*, *46*, *49*, *52*, *55*, *58*, *61*, *65*, *68*
mlr3::Measure, *37*, *70*
mlr3::mlr_learners, *4*, *6*, *8*, *10*, *12–14*, *16, 17*, *19–23*, *25–27*, *29, 30*, *32*, *34, 35*, *37*, *40*, *42*, *44*, *46*, *48, 49*, *51, 52*, *54, 55*, *57–59*, *61*, *64–66*, *68*, *71*
mlr3learners (mlr3learners-package), 3
mlr3learners-package, 3
mlr_learners_classif.cv_glmnet, 4, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.cv_glmnet(), *8*, *46*
mlr_learners_classif.glmnet, *7*, 7, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.kknn, *7*, *11*, 11, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.lda, *7*, *11*, *13*, 14, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.log_reg, *7*, *11*, *13*, *16*, 17, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.multinom, *7*, *11*, *13*, *16*, *19*, 20, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.naive_bayes, *7*, *11*, *13*, *16*, *19*, *21*, 22, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.nnet, *7*, *11*, *13*, *16*, *19*, *21*, *24*, 24, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.qda, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, 27, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.ranger, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, 30, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.svm, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, 33, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_classif.xgboost, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, 36, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_regr.cv_glmnet, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, 41, *48*, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_regr.cv_glmnet(), *8*, *46*
mlr_learners_regr.glmnet, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, 45, *51*, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_regr.kknn, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, 49, *54*, *57*, *60*, *64*, *66*, *72*
mlr_learners_regr.km, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, 52, *57*, *60*, *64*, *66*, *72*
mlr_learners_regr.lm, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, 55, *60*, *64*, *66*, *72*
mlr_learners_regr.nnet, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, 58, *64*, *66*, *72*
mlr_learners_regr.ranger, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*,

51, *54*, *57*, *60*, 60, *66*, *72*
mlr_learners_regr.svm, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, 65, *72*
mlr_learners_regr.xgboost, *7*, *11*, *13*, *16*, *19*, *21*, *24*, *26*, *29*, *33*, *35*, *40*, *44*, *48*, *51*, *54*, *57*, *60*, *64*, *66*, 67

nnet::multinom(), *20*
nnet::nnet.formula(), *24*, *58*

R6, *6*, *10*, *12*, *15*, *18*, *21*, *23*, *26*, *28*, *31*, *35*, *39*, *43*, *47*, *50*, *53*, *56*, *59*, *63*, *66*, *71*
ranger::ranger, *62*
ranger::ranger(), *30*

stats::glm(), *6*, *9*, *17*
stats::lm(), *55*

xgboost::predict.xgb.Booster(), *39*, *67*
xgboost::xgb.DMatrix, *39*, *67*
xgboost::xgb.importance(), *40*, *71*
xgboost::xgb.train(), *36*, *67*