

Package ‘mlr3learners’

March 18, 2021

Title Recommended Learners for 'mlr3'

Version 0.4.5

Description Recommended Learners for 'mlr3'. Extends 'mlr3' and 'mlr3proba' 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 R (>= 3.1.0), mlr3 (>= 0.11.0)

Imports data.table, mlr3misc (>= 0.5.0), paradox, R6

Suggests checkmate, DiceKriging, distr6, e1071, glmnet, kknn, knitr, lgr, MASS, mlr3proba (>= 0.2.5), nnet, pracma, ranger, rgenoud, rmarkdown, testthat (>= 3.0.0), xgboost

Encoding UTF-8

NeedsCompilation no

Config/testthat/edition 3

RoxygenNote 7.1.1

Author Michel Lang [cre, aut] (<<https://orcid.org/0000-0001-9754-0393>>),
Quay Au [aut] (<<https://orcid.org/0000-0002-5252-8902>>),
Stefan Coors [aut] (<<https://orcid.org/0000-0002-7465-2146>>),
Patrick Schratz [aut] (<<https://orcid.org/0000-0003-0748-6624>>)

Maintainer Michel Lang <michellang@gmail.com>

Repository CRAN

Date/Publication 2021-03-18 13:20:03 UTC

R topics documented:

mlr3learners-package	2
mlr_learners_classif.cv_glmnet	3
mlr_learners_classif.glmnet	4
mlr_learners_classif.kknn	5
mlr_learners_classif.lda	7
mlr_learners_classif.log_reg	8
mlr_learners_classif.multinom	10
mlr_learners_classif.naive_bayes	11
mlr_learners_classif.nnet	12
mlr_learners_classif.qda	13
mlr_learners_classif.ranger	14
mlr_learners_classif.svm	16
mlr_learners_classif.xgboost	17
mlr_learners_regr.cv_glmnet	19
mlr_learners_regr.glmnet	20
mlr_learners_regr.kknn	21
mlr_learners_regr.km	23
mlr_learners_regr.lm	24
mlr_learners_regr.ranger	26
mlr_learners_regr.svm	27
mlr_learners_regr.xgboost	29
mlr_learners_surv.cv_glmnet	30
mlr_learners_surv.glmnet	32
mlr_learners_surv.ranger	33
mlr_learners_surv.xgboost	35

Index	37
--------------	-----------

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: Michel Lang <micHELLang@gmail.com> ([ORCID](#))

Authors:

- Quay Au <quayau@gmail.com> ([ORCID](#))
- Stefan Coors <mail@stefancoors.de> ([ORCID](#))
- Patrick Schratz <patrick.schratz@gmail.com> ([ORCID](#))

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.

Dictionary

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

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

Super classes

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

Methods**Public methods:**

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

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

Usage:

```
LearnerClassifCVGlmnet$new()
```

Method `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](https://doi.org/10.18637/jss.v033.i01).

See Also

Dictionary of Learners: [mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("glmnet")) {
  learner = mlr3::lrn("classif.cv_glmnet")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

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**.

Caution: This learner is different to `_glmnet` in that it does not use the internal optimization of lambda. The parameter needs to be tuned by the user. Essentially, one needs to tune parameter `s` which is used at predict-time.

See <https://stackoverflow.com/questions/50995525/> for more information.

Dictionary

This **Learner** can be instantiated via the dictionary [mlr_learners](#) or with the associated sugar function `lrn()`:

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

Super classes

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

Methods

Public methods:

- [LearnerClassifGlmnet\\$new\(\)](#)
- [LearnerClassifGlmnet\\$clone\(\)](#)

Method `new()`: Creates a new instance of this [R6](#) class.

Usage:

```
LearnerClassifGlmnet$new()
```

Method `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](https://doi.org/10.18637/jss.v033.i01).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("glmnet")) {  
  learner = mlr3::lrn("classif.glmnet")  
  print(learner)  
  
  # available parameters:  
  learner$param_set$ids()  
}
```

mlr_learners_classif.kknn

k-Nearest-Neighbor Classification Learner

Description

k-Nearest-Neighbor classification. Calls [kknn::kknn\(\)](#) from package [kknn](#).

Dictionary

This [Learner](#) can be instantiated via the dictionary [mlr_learners](#) or with the associated sugar function [lrn\(\)](#):

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

Super classes

```
mlr3::Learner -> mlr3::LearnerClassif -> LearnerClassifKknn
```

Methods

Public methods:

- [LearnerClassifKknn\\$new\(\)](#)
- [LearnerClassifKknn\\$clone\(\)](#)

Method [new\(\)](#): Creates a new instance of this [R6](#) class.

Usage:

```
LearnerClassifKknn$new()
```

Method [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()`.
- `pars`: Training parameters for calling [kknn::kknn\(\)](#) during `$predict()`.
- `kknn`: Model as returned by [kknn::kknn\(\)](#), only available **after** `$predict()` has been called.

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](https://doi.org/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](https://doi.org/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](https://doi.org/10.1109/TIT.1967.1053964).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("kknn")) {  
  learner = mlr3::lrn("classif.kknn")  
  print(learner)  
  
  # available parameters:  
  learner$param_set$ids()  
}
```

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 [Learner](#) can be instantiated via the [dictionary mlr_learners](#) or with the associated sugar function `lrn()`:

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

Super classes

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

Methods**Public methods:**

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

Method `new()`: Creates a new instance of this [R6](#) class.

Usage:

```
LearnerClassifLDA$new()
```

Method `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

Dictionary of Learners: [mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("MASS")) {
  learner = mlr3::lrn("classif.lda")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

```
mlr_learners_classif.log_reg
```

Logistic Regression Classification Learner

Description

Classification via logistic regression. Calls `stats::glm()` with family set to "binomial".

Custom mlr3 defaults

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

Dictionary

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

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


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()`

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

Usage:

```
LearnerClassifLogReg$new()
```

Method `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

Dictionary of Learners: [mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("stats")) {
  learner = mlr3::lrn("classif.log_reg")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

```
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 [Learner](#) can be instantiated via the [dictionary mlr_learners](#) or with the associated sugar function `lrn()`:

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

Super classes

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

Methods

Public methods:

- [LearnerClassifMultinom\\$new\(\)](#)
- [LearnerClassifMultinom\\$clone\(\)](#)

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

Usage:

```
LearnerClassifMultinom$new()
```

Method `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

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("nnet")) {
  learner = mlr3::lrn("classif.multinom")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

```
mlr_learners_classif.naive_bayes
```

Naive Bayes Classification Learner

Description

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

Dictionary

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

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

Super classes

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

Methods

Public methods:

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

Method `new()`: Creates a new instance of this [R6](#) class.

Usage:

```
LearnerClassifNaiveBayes$new()
```

Method `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

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("e1071")) {
  learner = mlr3::lrn("classif.naive_bayes")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

```
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 **mlr3keras**.

Dictionary

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

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

Custom mlr3 defaults

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

Super classes

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

Methods

Public methods:

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

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

Usage:

```
LearnerClassifNnet$new()
```

Method `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](https://doi.org/10.1017/cbo9780511812651).

See Also

Dictionary of Learners: [mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("nnet")) {  
  learner = mlr3::lrn("classif.nnet")  
  print(learner)  
  
  # available parameters:  
  learner$param_set$ids()  
}
```

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 **Learner** can be instantiated via the dictionary [mlr_learners](#) or with the associated sugar function `lrn()`:

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

Super classes

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

Methods

Public methods:

- [LearnerClassifQDA\\$new\(\)](#)
- [LearnerClassifQDA\\$clone\(\)](#)

Method `new()`: Creates a new instance of this [R6](#) class.

Usage:

```
LearnerClassifQDA$new()
```

Method `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

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("MASS")) {  
  learner = mlr3::lrn("classif.qda")  
  print(learner)  
  
  # available parameters:  
  learner$param_set$ids()  
}
```

mlr_learners_classif.ranger

Ranger Classification Learner

Description

Random classification forest. Calls [ranger::ranger\(\)](#) from package **ranger**.

Custom mlr3 defaults

- `num.threads`:
 - Actual default: NULL, triggering auto-detection of the number of CPUs.
 - Adjusted value: 1.
 - Reason for change: Conflicting with parallelization via **future**.

Dictionary

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

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

Super classes

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

Methods**Public methods:**

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

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

Usage:

```
LearnerClassifRanger$new()
```

Method `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().

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

Usage:

```
LearnerClassifRanger$oob_error()
```

Returns: numeric(1).

Method `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](https://doi.org/10.18637/jss.v077.i01).

Breiman, Leo (2001). "Random Forests." *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565, doi: [10.1023/A:1010933404324](https://doi.org/10.1023/A:1010933404324).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("ranger")) {
  learner = mlr3::lrn("classif.ranger")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

mlr_learners_classif.svm

Support Vector Machine

Description

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

Dictionary

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

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

Super classes

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

Methods**Public methods:**

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

Method `new()`: Creates a new instance of this [R6](#) class.

Usage:

```
LearnerClassifSVM$new()
```

Method `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](https://doi.org/10.1007/BF00994018).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("e1071")) {  
  learner = mlr3::lrn("classif.svm")  
  print(learner)  
  
  # available parameters:  
  learner$param_set$ids()  
}
```

mlr_learners_classif.xgboost

Extreme Gradient Boosting Classification Learner

Description

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

Custom mlr3 defaults

- nrounds:
 - Actual default: no default.
 - Adjusted default: 1.
 - Reason for change: Without a default construction of the learner would error. Just setting a nonsense default to workaround this. nrounds needs to be tuned by the user.
- 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.

Dictionary

This [Learner](#) can be instantiated via the dictionary [mlr_learners](#) or with the associated sugar function [lrn\(\)](#):

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

Super classes

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

Methods

Public methods:

- [LearnerClassifXgboost\\$new\(\)](#)
- [LearnerClassifXgboost\\$importance\(\)](#)
- [LearnerClassifXgboost\\$clone\(\)](#)

Method [new\(\)](#): Creates a new instance of this [R6](#) class.

Usage:

```
LearnerClassifXgboost$new()
```

Method [importance\(\)](#): The importance scores are calculated with [xgboost::xgb.importance\(\)](#).

Usage:

```
LearnerClassifXgboost$importance()
```

Returns: Named numeric().

Method [clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

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](https://doi.org/10.1145/2939672.2939785).

See Also

Dictionary of Learners: [mlr3::mlr_learners](#)

Examples

```

if (requireNamespace("xgboost")) {
  learner = mlr3::lrn("classif.xgboost")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}

```

```
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**.

The default for hyperparameter family is set to "gaussian".

Dictionary

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

```

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

```

Super classes

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

Methods**Public methods:**

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

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

Usage:

```
LearnerRegrCVGlmnet$new()
```

Method `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](https://doi.org/10.18637/jss.v033.i01).

See Also

Dictionary of Learners: [mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("glmnet")) {
  learner = mlr3::lrn("regr.cv_glmnet")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

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**.

The default for hyperparameter family is set to "gaussian".

Caution: This learner is different to `cv_glmnet` in that it does not use the internal optimization of lambda. The parameter needs to be tuned by the user. Essentially, one needs to tune parameter `s` which is used at predict-time.

See <https://stackoverflow.com/questions/50995525/> for more information.

Dictionary

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

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

Super classes

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

Methods

Public methods:

- [LearnerRegrGlmnet\\$new\(\)](#)
- [LearnerRegrGlmnet\\$clone\(\)](#)

Method `new()`: Creates a new instance of this [R6](#) class.

Usage:

```
LearnerRegrGlmnet$new()
```

Method `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](https://doi.org/10.18637/jss.v033.i01).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("glmnet")) {  
  learner = mlr3::lrn("regr.glmnet")  
  print(learner)  
  
  # available parameters:  
  learner$param_set$ids()  
}
```

mlr_learners_regr.kknn

k-Nearest-Neighbor Regression Learner

Description

k-Nearest-Neighbor regression. Calls [kknn::kknn\(\)](#) from package [kknn](#).

Dictionary

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

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

Super classes

```
mlr3::Learner -> mlr3::LearnerRegr -> LearnerRegrKknn
```

Methods

Public methods:

- `LearnerRegrKknn$new()`
- `LearnerRegrKknn$clone()`

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

Usage:

```
LearnerRegrKknn$new()
```

Method `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()`.
- `pars`: Training parameters for calling `kknn::kknn()` during `$predict()`.
- `kknn`: Model as returned by `kknn::kknn()`, only available **after** `$predict()` has been called.

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](https://doi.org/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](https://doi.org/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](https://doi.org/10.1109/TIT.1967.1053964).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("kknn")) {
  learner = mlr3::lrn("regr.kknn")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

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 [Learner](#) can be instantiated via the [dictionary mlr_learners](#) or with the associated sugar function [lrn\(\)](#):

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

Super classes

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

Methods**Public methods:**

- [LearnerRegrKM\\$new\(\)](#)
- [LearnerRegrKM\\$clone\(\)](#)

Method `new()`: Creates a new instance of this [R6](#) class.

Usage:

```
LearnerRegrKM$new()
```

Method `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](https://doi.org/10.18637/jss.v051.i01).

See Also

Dictionary of Learners: [mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("DiceKriging")) {
  learner = mlr3::lrn("regr.km")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

mlr_learners_regr.lm *Linear Model Regression Learner*

Description

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

Dictionary

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

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


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()`

Method `new()`: Creates a new instance of this [R6](#) class.

Usage:

```
LearnerRegrLM$new()
```

Method `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

Dictionary of Learners: [mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("stats")) {
  learner = mlr3::lrn("regr.lm")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

```
mlr_learners_regr.ranger
```

Ranger Regression Learner

Description

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

Dictionary

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

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

Custom mlr3 defaults

- num.threads:
 - Actual default: NULL, triggering auto-detection of the number of CPUs.
 - Adjusted value: 1.
 - Reason for change: Conflicting with parallelization via **future**.

Super classes

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

Methods

Public methods:

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

Method `new()`: Creates a new instance of this [R6](#) class.

Usage:

```
LearnerRegrRanger$new()
```

Method `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().

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

Usage:

```
LearnerRegrRanger$oob_error()
```

Returns: `numeric(1)`.

Method `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](https://doi.org/10.18637/jss.v077.i01).

Breiman, Leo (2001). “Random Forests.” *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565, doi: [10.1023/A:1010933404324](https://doi.org/10.1023/A:1010933404324).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("ranger")) {
  learner = mlr3::lrn("regr.ranger")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

`mlr_learners_regr.svm` *Support Vector Machine*

Description

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

Dictionary

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

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

Super classes

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

Methods

Public methods:

- [LearnerRegrSVM\\$new\(\)](#)
- [LearnerRegrSVM\\$clone\(\)](#)

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

Usage:

```
LearnerRegrSVM$new()
```

Method `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](https://doi.org/10.1007/BF00994018).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("e1071")) {  
  learner = mlr3::lrn("regr.svm")  
  print(learner)  
  
  # available parameters:  
  learner$param_set$ids()  
}
```

 mlr_learners_regr.xgboost

Extreme Gradient Boosting Regression Learner

Description

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

Dictionary

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

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

Custom mlr3 defaults

- nrounds:
 - Actual default: no default.
 - Adjusted default: 1.
 - Reason for change: Without a default construction of the learner would error. Just setting a nonsense default to workaround this. nrounds needs to be tuned by the user.
- 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.

Super classes

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

Methods

Public methods:

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

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

Usage:

```
LearnerRegrXgboost$new()
```

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

Usage:

```
LearnerRegrXgboost$importance()
```

Returns: Named numeric().

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

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](https://doi.org/10.1145/2939672.2939785).

See Also

Dictionary of Learners: [mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("xgboost")) {
  learner = mlr3::lrn("regr.xgboost")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

mlr_learners_surv.cv_glmnet

Cross-Validated GLM with Elastic Net Regularization Survival Learner

Description

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

The default for hyperparameter family is set to "cox".

Dictionary

This [Learner](#) can be instantiated via the [dictionary mlr_learners](#) or with the associated sugar function [lrn\(\)](#):

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

Super classes

```
mlr3::Learner -> mlr3proba::LearnerSurv -> LearnerSurvCVGlmnet
```

Methods**Public methods:**

- [LearnerSurvCVGlmnet\\$new\(\)](#)
- [LearnerSurvCVGlmnet\\$clone\(\)](#)

Method [new\(\)](#): Creates a new instance of this [R6](#) class.

Usage:

```
LearnerSurvCVGlmnet$new()
```

Method [clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
LearnerSurvCVGlmnet$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](https://doi.org/10.18637/jss.v033.i01).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("glmnet")) {
  learner = mlr3::lrn("surv.cv_glmnet")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

`mlr_learners_surv.glmnet`*GLM with Elastic Net Regularization Survival Learner*

Description

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

The default for hyperparameter family is set to "cox".

Caution: This learner is different to `cv.glmnet` in that it does not use the internal optimization of lambda. The parameter needs to be tuned by the user. Essentially, one needs to tune parameter `s` which is used at predict-time.

See <https://stackoverflow.com/questions/50995525/> for more information.

Dictionary

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

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

Super classes

```
mlr3::Learner -> mlr3proba::LearnerSurv -> LearnerSurvGlmnet
```

Methods

Public methods:

- `LearnerSurvGlmnet$new()`
- `LearnerSurvGlmnet$clone()`

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

Usage:

```
LearnerSurvGlmnet$new()
```

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

Usage:

```
LearnerSurvGlmnet$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](https://doi.org/10.18637/jss.v033.i01).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("glmnet")) {
  learner = mlr3::lrn("surv.glmnet")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

```
mlr_learners_surv.ranger
```

Ranger Survival Learner

Description

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

Dictionary

This **Learner** can be instantiated via the [dictionary mlr_learners](#) or with the associated sugar function `lrn()`:

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

Custom mlr3 defaults

- num. threads:
 - Actual default: NULL, triggering auto-detection of the number of CPUs.
 - Adjusted value: 1.
 - Reason for change: Conflicting with parallelization via **future**.

Super classes

```
mlr3::Learner -> mlr3proba::LearnerSurv -> LearnerSurvRanger
```

Methods**Public methods:**

- `LearnerSurvRanger$new()`
- `LearnerSurvRanger$importance()`
- `LearnerSurvRanger$oob_error()`

- [LearnerSurvRanger\\$clone\(\)](#)

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

Usage:

```
LearnerSurvRanger$new()
```

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

Usage:

```
LearnerSurvRanger$importance()
```

Returns: Named numeric().

Method `oob_error()`: The out-of-bag error is extracted from the model slot `prediction.error`.

Usage:

```
LearnerSurvRanger$oob_error()
```

Returns: numeric(1).

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

Usage:

```
LearnerSurvRanger$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](https://doi.org/10.18637/jss.v077.i01).

Breiman, Leo (2001). “Random Forests.” *Machine Learning*, **45**(1), 5–32. ISSN 1573-0565, doi: [10.1023/A:1010933404324](https://doi.org/10.1023/A:1010933404324).

See Also

Dictionary of Learners: [mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("ranger")) {
  learner = mlr3::lrn("surv.ranger")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```

 mlr_learners_surv.xgboost

Extreme Gradient Boosting Survival Learner

Description

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

Custom mlr3 defaults

- nrounds:
 - Actual default: no default.
 - Adjusted default: 1.
 - Reason for change: Without a default construction of the learner would error. Just setting a nonsense default to workaround this. nrounds needs to be tuned by the user.
- 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.
- objective:
 - Actual default: `reg:squarederror`
 - Adjusted default: `survival:cox`
 - Reason for change: Changed to a survival objective.

Dictionary

This **Learner** can be instantiated via the [dictionary mlr_learners](#) or with the associated sugar function `lrn()`:

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

Super classes

```
mlr3::Learner -> mlr3proba::LearnerSurv -> LearnerSurvXgboost
```

Methods

Public methods:

- [LearnerSurvXgboost\\$new\(\)](#)
- [LearnerSurvXgboost\\$importance\(\)](#)
- [LearnerSurvXgboost\\$clone\(\)](#)

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

Usage:

```
LearnerSurvXgboost$new()
```

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

Usage:

```
LearnerSurvXgboost$importance()
```

Returns: Named numeric().

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

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](https://doi.org/10.1145/2939672.2939785).

See Also

[Dictionary of Learners: mlr3::mlr_learners](#)

Examples

```
if (requireNamespace("xgboost")) {  
  learner = mlr3::lrn("surv.xgboost")  
  print(learner)  
  
  # available parameters:  
  learner$param_set$ids()  
}
```

Index

`contr.poly()`, [9, 25](#)
`contr.treatment()`, [9, 25](#)

`DiceKriging::km()`, [23](#)
Dictionary, [4, 5, 7–11, 13, 14, 16–18, 20, 21, 23–25, 27, 28, 30, 31, 33, 34, 36](#)
dictionary, [3, 4, 6–8, 10–13, 15, 16, 18–20, 22–24, 26, 27, 29, 31–33, 35](#)

`e1071::naiveBayes()`, [11](#)
`e1071::svm()`, [16, 27](#)

`glmnet::cv.glmnet()`, [3, 19, 30](#)
`glmnet::glmnet()`, [4, 20, 32](#)

`kknn::kknn()`, [5, 6, 21, 22](#)

Learner, [3, 4, 6–8, 10–13, 15, 16, 18–20, 22–24, 26, 27, 29, 31–33, 35](#)

LearnerClassifCVGlmnet
(`mlr_learners_classif.cv_glmnet`), [3](#)

LearnerClassifGlmnet
(`mlr_learners_classif.glmnet`), [4](#)

LearnerClassifKknn
(`mlr_learners_classif.kknn`), [5](#)

LearnerClassifLDA
(`mlr_learners_classif.lda`), [7](#)

LearnerClassifLogReg
(`mlr_learners_classif.log_reg`), [8](#)

LearnerClassifMultinom
(`mlr_learners_classif.multinom`), [10](#)

LearnerClassifNaiveBayes
(`mlr_learners_classif.naive_bayes`), [11](#)

LearnerClassifNnet
(`mlr_learners_classif.nnet`), [12](#)

LearnerClassifQDA
(`mlr_learners_classif.qda`), [13](#)

LearnerClassifRanger
(`mlr_learners_classif.ranger`), [14](#)

LearnerClassifSVM
(`mlr_learners_classif.svm`), [16](#)

LearnerClassifXgboost
(`mlr_learners_classif.xgboost`), [17](#)

LearnerRegrCVGlmnet
(`mlr_learners_regr.cv_glmnet`), [19](#)

LearnerRegrGlmnet
(`mlr_learners_regr.glmnet`), [20](#)

LearnerRegrKknn
(`mlr_learners_regr.kknn`), [21](#)

LearnerRegrKM(`mlr_learners_regr.km`), [23](#)

LearnerRegrLM(`mlr_learners_regr.lm`), [24](#)

LearnerRegrRanger
(`mlr_learners_regr.ranger`), [26](#)

LearnerRegrSVM(`mlr_learners_regr.svm`), [27](#)

LearnerRegrXgboost
(`mlr_learners_regr.xgboost`), [29](#)

Learners, [4, 5, 7–11, 13, 14, 16–18, 20, 21, 23–25, 27, 28, 30, 31, 33, 34, 36](#)

LearnerSurvCVGlmnet
(`mlr_learners_surv.cv_glmnet`), [30](#)

LearnerSurvGlmnet
(`mlr_learners_surv.glmnet`), [32](#)

LearnerSurvRanger
(`mlr_learners_surv.ranger`), [33](#)

LearnerSurvXgboost
(`mlr_learners_surv.xgboost`), [35](#)

`lrn()`, [3, 4, 6–8, 10–13, 15, 16, 18–20, 22–24, 26, 27, 29, 31–33, 35](#)

MASS::lda(), [7](#)

MASS::qda(), 13
 mlr3::Learner, 3, 4, 6, 7, 9–13, 15, 16,
 18–20, 22, 23, 25, 26, 28, 29, 31–33,
 35
 mlr3::LearnerClassif, 3, 4, 6, 7, 9–13, 15,
 16, 18
 mlr3::LearnerRegr, 19, 20, 22, 23, 25, 26,
 28, 29
 mlr3::mlr_learners, 4, 5, 7–11, 13, 14,
 16–18, 20, 21, 23–25, 27, 28, 30, 31,
 33, 34, 36
 mlr3learners (mlr3learners-package), 2
 mlr3learners-package, 2
 mlr3proba::LearnerSurv, 31–33, 35
 mlr_learners, 3, 4, 6–8, 10–13, 15, 16,
 18–20, 22–24, 26, 27, 29, 31–33, 35
 mlr_learners_classif.cv_glmnet, 3
 mlr_learners_classif.glmnet, 4
 mlr_learners_classif.kknn, 5
 mlr_learners_classif.lda, 7
 mlr_learners_classif.log_reg, 8
 mlr_learners_classif.multinom, 10
 mlr_learners_classif.naive_bayes, 11
 mlr_learners_classif.nnet, 12
 mlr_learners_classif.qda, 13
 mlr_learners_classif.ranger, 14
 mlr_learners_classif.svm, 16
 mlr_learners_classif.xgboost, 17
 mlr_learners_regr.cv_glmnet, 19
 mlr_learners_regr.glmnet, 20
 mlr_learners_regr.kknn, 21
 mlr_learners_regr.km, 23
 mlr_learners_regr.lm, 24
 mlr_learners_regr.ranger, 26
 mlr_learners_regr.svm, 27
 mlr_learners_regr.xgboost, 29
 mlr_learners_surv.cv_glmnet, 30
 mlr_learners_surv.glmnet, 32
 mlr_learners_surv.ranger, 33
 mlr_learners_surv.xgboost, 35

 nnet::multinom(), 10
 nnet::nnet.formula(), 12

 R6, 3, 5–7, 9–12, 14–16, 18, 19, 21–23, 25, 26,
 28, 29, 31, 32, 34, 36
 ranger::ranger(), 14, 26, 33

 stats::glm(), 8

 stats::lm(), 24

 xgboost::xgb.importance(), 18, 30, 36
 xgboost::xgb.train(), 17, 29, 35