

# Package ‘mlr3learners’

April 22, 2020

**Title** Recommended Learners for 'mlr3'

**Version** 0.2.0

**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** R (>= 3.1.0)

**Imports** data.table, mlr3 (>= 0.2.0), mlr3misc (>= 0.2.0), paradox, R6

**Suggests** bibtext, checkmate, DiceKriging, e1071, glmnet, kkn, knitr, lgr, MASS, nnet, ranger, rmarkdown, testthat, xgboost

**RdMacros** mlr3misc

**Encoding** UTF-8

**NeedsCompilation** no

**RoxygenNote** 7.1.0

**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** 2020-04-22 20:54:13 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 . . . . .	9
mlr_learners_classif.naive_bayes . . . . .	10
mlr_learners_classif.qda . . . . .	11
mlr_learners_classif.ranger . . . . .	13
mlr_learners_classif.svm . . . . .	14
mlr_learners_classif.xgboost . . . . .	15
mlr_learners_regr.cv_glmnet . . . . .	17
mlr_learners_regr.glmnet . . . . .	18
mlr_learners_regr.kknn . . . . .	19
mlr_learners_regr.km . . . . .	21
mlr_learners_regr.lm . . . . .	22
mlr_learners_regr.ranger . . . . .	23
mlr_learners_regr.svm . . . . .	25
mlr_learners_regr.xgboost . . . . .	26
<b>Index</b>	<b>28</b>

---

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

---

### Description

More learners are available in the mlr3learners repository on Github (<https://github.com/mlr3learners>). There also is a wiki page listing all currently available custom learners (<https://github.com/mlr-org/mlr3learners/wiki/Extra-Learners>). A guide on how to create custom learners is covered in the book: <https://mlr3book.ml-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**.

**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.

**References**

Hechenbichler K, Schliep K (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 RJ (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 T, Hart P (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". Argument model is set to FALSE.

## 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.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**.

**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 MN, Ziegler A (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 L (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**

A learner for a classification support vector machine implemented in [e1071::svm\(\)](#).

**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 C, Vapnik V (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.
- `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 T, Guestrin C (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 changed 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 changed 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.

## References

Hechenbichler K, Schliep K (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 RJ (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 T, Hart P (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")
```

**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 MN, Ziegler A (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 L (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

A learner for a regression support vector machine implemented in [e1071::svm\(\)](#).

## 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 C, Vapnik V (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**.

## 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.
- 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("regr.xgboost")
lrn("regr.xgboost")
```

**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 T, Guestrin C (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()
}
```

# Index

`contr.poly()`, 8, 22  
`contr.treatment()`, 8, 22

`DiceKriging::km()`, 21  
Dictionary, 4–6, 8–12, 14–16, 18–20, 22–24, 26, 27  
dictionary, 3, 4, 6–10, 12–14, 16–18, 20–23, 25, 26

`e1071::naiveBayes()`, 10  
`e1071::svm()`, 14, 25

`glmnet::cv.glmnet()`, 3, 17  
`glmnet::glmnet()`, 4, 18

`kknn::kknn()`, 5, 19

Learner, 3, 4, 6–10, 12–14, 16–18, 20–23, 25, 26

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`), 9

LearnerClassifNaiveBayes  
(`mlr_learners_classif.naive_bayes`), 10

LearnerClassifQDA  
(`mlr_learners_classif.qda`), 11

LearnerClassifRanger  
(`mlr_learners_classif.ranger`), 13

LearnerClassifSVM  
(`mlr_learners_classif.svm`), 14

LearnerClassifXgboost  
(`mlr_learners_classif.xgboost`), 15

LearnerRegrCVGlmnet  
(`mlr_learners_regr.cv_glmnet`), 17

LearnerRegrGlmnet  
(`mlr_learners_regr.glmnet`), 18

LearnerRegrKknn  
(`mlr_learners_regr.kknn`), 19

LearnerRegrKM (`mlr_learners_regr.km`), 21

LearnerRegrLM (`mlr_learners_regr.lm`), 22

LearnerRegrRanger  
(`mlr_learners_regr.ranger`), 23

LearnerRegrSVM (`mlr_learners_regr.svm`), 25

LearnerRegrXgboost  
(`mlr_learners_regr.xgboost`), 26

Learners, 4–6, 8–12, 14–16, 18–20, 22–24, 26, 27

`lrn()`, 3, 4, 6–10, 12–14, 16–18, 20–23, 25, 26

MASS::lda(), 7  
MASS::qda(), 11

`mlr3::Learner`, 3, 4, 6, 7, 9–14, 16–18, 20, 21, 23–25, 27  
`mlr3::LearnerClassif`, 3, 4, 6, 7, 9–14, 16  
`mlr3::LearnerRegr`, 17, 18, 20, 21, 23–25, 27  
`mlr3::mlr_learners`, 4–6, 8–12, 14–16, 18–20, 22–24, 26, 27

`mlr3learners` (`mlr3learners-package`), 2  
`mlr3learners-package`, 2  
`mlr_learners`, 3, 4, 6–10, 12–14, 16–18, 20–23, 25, 26  
`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`, 9  
`mlr_learners_classif.naive_bayes`, 10  
`mlr_learners_classif.qda`, 11  
`mlr_learners_classif.ranger`, 13  
`mlr_learners_classif.svm`, 14  
`mlr_learners_classif.xgboost`, 15  
`mlr_learners_regr.cv_glmnet`, 17  
`mlr_learners_regr.glmnet`, 18  
`mlr_learners_regr.kknn`, 19  
`mlr_learners_regr.km`, 21  
`mlr_learners_regr.lm`, 22  
`mlr_learners_regr.ranger`, 23  
`mlr_learners_regr.svm`, 25  
`mlr_learners_regr.xgboost`, 26

`nnet::multinom()`, 9

R6, 3, 5–7, 9–14, 16, 17, 19–21, 23–25, 27  
`ranger::ranger()`, 13, 23

`stats::glm()`, 8  
`stats::lm()`, 22

`xgboost::xgb.importance()`, 16, 27  
`xgboost::xgb.train()`, 15, 26