

# Package ‘mlr3learners’

January 6, 2023

**Title** Recommended Learners for ‘mlr3’

**Version** 0.5.6

**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 (>= 0.14.1), R (>= 3.1.0)

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

**Suggests** DiceKriging, e1071, glmnet, kknn, knitr, lgr, MASS, nnet, pracma, ranger, rgenoud, rmarkdown, testthat (>= 3.0.0), xgboost (>= 1.6.0)

**Config/testthat/edition** 3

**Encoding** UTF-8

**NeedsCompilation** no

**RoxygenNote** 7.2.3

**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' 'zzz.R'

**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** 2023-01-06 22:30:21 UTC

## R topics documented:

mlr3learners-package . . . . .	2
mlr_learners_classif.cv_glmnet . . . . .	3
mlr_learners_classif.glmnet . . . . .	6
mlr_learners_classif.kknn . . . . .	9
mlr_learners_classif.lda . . . . .	12
mlr_learners_classif.log_reg . . . . .	14
mlr_learners_classif.multinom . . . . .	16
mlr_learners_classif.naive_bayes . . . . .	19
mlr_learners_classif.nnet . . . . .	21
mlr_learners_classif.qda . . . . .	23
mlr_learners_classif.ranger . . . . .	26
mlr_learners_classif.svm . . . . .	29
mlr_learners_classif.xgboost . . . . .	31
mlr_learners_regr.cv_glmnet . . . . .	36
mlr_learners_regr.glmnet . . . . .	39
mlr_learners_regr.kknn . . . . .	42
mlr_learners_regr.km . . . . .	45
mlr_learners_regr.lm . . . . .	47
mlr_learners_regr.nnet . . . . .	50
mlr_learners_regr.ranger . . . . .	52
mlr_learners_regr.svm . . . . .	55
mlr_learners_regr.xgboost . . . . .	57
<b>Index</b>	<b>63</b>

---

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")
```

**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
alignment	character	lambda	lambda, fraction	-
alpha	numeric	1		[0, 1]
big	numeric	9.9e+35		$(-\infty, \infty)$
devmax	numeric	0.999		[0, 1]
dfmax	integer	-		[0, $\infty$ )
epsnr	numeric	1e-08		[0, 1]
eps	numeric	1e-06		[0, 1]
exclude	integer	-		[1, $\infty$ )
exmx	numeric	250		$(-\infty, \infty)$
fdev	numeric	1e-05		[0, 1]
foldid	untyped			-
gamma	untyped	-		-
grouped	logical	TRUE	TRUE, FALSE	-
intercept	logical	TRUE	TRUE, FALSE	-
keep	logical	FALSE	TRUE, FALSE	-
lambda.min.ratio	numeric	-		[0, 1]
lambda	untyped	-		-
lower.limits	untyped	-		-
maxit	integer	100000		[1, $\infty$ )
mnlam	integer	5		[1, $\infty$ )
mxitnr	integer	25		[1, $\infty$ )
mxit	integer	100		[1, $\infty$ )
nfolds	integer	10		[3, $\infty$ )
nlambda	integer	100		[1, $\infty$ )
offset	untyped			-
parallel	logical	FALSE	TRUE, FALSE	-
penalty.factor	untyped	-		-
pmax	integer	-		[0, $\infty$ )
pmin	numeric	1e-09		[0, 1]
prec	numeric	1e-10		$(-\infty, \infty)$
predict.gamma	numeric	gamma.1se		$(-\infty, \infty)$
relax	logical	FALSE	TRUE, FALSE	-
s	numeric	lambda.1se		[0, $\infty$ )
standardize	logical	TRUE	TRUE, FALSE	-
standardize.response	logical	FALSE	TRUE, FALSE	-
thresh	numeric	1e-07		[0, $\infty$ )
trace.it	integer	0		[0, 1]
type.gaussian	character	-	covariance, naive	-
type.logistic	character	-	Newton, modified.Newton	-
type.measure	character	deviance	deviance, class, auc, mse, mae	-
type.multinomial	character	-	ungrouped, grouped	-
upper.limits	untyped	-		-

## 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()`

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

*Usage:*

```
LearnerClassifCVGlmnet$new()
```

**Method** `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.

**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.

## See Also

- Chapter in the **mlr3book**: <https://mlr3book.mlr-org.com/basics.html#learners>
- Package **mlr3extralearners** for more learners.
- Dictionary of Learners: [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

```
if (requireNamespace("glmnet", quietly = TRUE)) {
  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**.

## 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 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 [Learner](#) can be instantiated via the [dictionary mlr\\_learners](#) or with the associated sugar function `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]
big	numeric	9.9e+35		$(-\infty, \infty)$
devmax	numeric	0.999		[0, 1]
dfmax	integer	-		[0, $\infty$ )
eps	numeric	1e-06		[0, 1]
epsnr	numeric	1e-08		[0, 1]
exact	logical	FALSE	TRUE, FALSE	-
exclude	integer	-		[1, $\infty$ )
exmx	numeric	250		$(-\infty, \infty)$
fdev	numeric	1e-05		[0, 1]
gamma	numeric	1		$(-\infty, \infty)$
intercept	logical	TRUE	TRUE, FALSE	-
lambda	untyped	-		-
lambda.min.ratio	numeric	-		[0, 1]
lower.limits	untyped	-		-
maxit	integer	100000		[1, $\infty$ )
mnlam	integer	5		[1, $\infty$ )
mxit	integer	100		[1, $\infty$ )
mxitnr	integer	25		[1, $\infty$ )
nlambda	integer	100		[1, $\infty$ )
newoffset	untyped	-		-
offset	untyped	-		-
penalty.factor	untyped	-		-
pmax	integer	-		[0, $\infty$ )
pmin	numeric	1e-09		[0, 1]
prec	numeric	1e-10		$(-\infty, \infty)$
relax	logical	FALSE	TRUE, FALSE	-
s	numeric	0.01		[0, $\infty$ )
standardize	logical	TRUE	TRUE, FALSE	-
standardize.response	logical	FALSE	TRUE, FALSE	-

thresh	numeric	1e-07		[0, ∞)
trace.it	integer	0		[0, 1]
type.gaussian	character	-	covariance, naive	-
type.logistic	character	-	Newton, modified.Newton	-
type.multinomial	character	-	ungrouped, grouped	-
upper.limits	untyped	-		-

### 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 -> LearnerClassifGlmnet
```

### Methods

#### Public methods:

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

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

*Usage:*

```
LearnerClassifGlmnet$new()
```

**Method** `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.

**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.

## See Also

- Chapter in the **mlr3book**: <https://mlr3book.mlr-org.com/basics.html#learners>
- Package **mlr3extralearners** for more learners.
- **Dictionary of Learners**: [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

```
if (requireNamespace("glmnet", quietly = TRUE)) {
  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**.

## Initial parameter values

- `store_model`:
  - See note.

## 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")
```

## 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		
store_model	logical	FALSE	TRUE, FALSE

## 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()`.
- `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/basics.html#learners>
- Package `mlr3extralearners` for more learners.
- [Dictionary of Learners: 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**

```

if (requireNamespace("kknn", quietly = TRUE)) {
  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")

```

**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()`

**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

- Chapter in the `mlr3book`: <https://mlr3book.ml-org.com/basics.html#learners>
- Package `mlr3extralearners` for more learners.
- *Dictionary of Learners*: `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, `mlr3tuningpaces` 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**

```
if (requireNamespace("MASS", quietly = TRUE)) {
  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".

**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.

**Weights**

It is not advisable to change the weights of a logistic regression. For more details, see this question on [Cross Validated](#).

**Initial parameter values**

- 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")
```

**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	-		-
epsilon	numeric	1e-08		$(-\infty, \infty)$
etastart	untyped	-		-
maxit	numeric	25		$(-\infty, \infty)$
model	logical	TRUE	TRUE, FALSE	-
mustart	untyped	-		-
offset	untyped	-		-
singular.ok	logical	TRUE	TRUE, FALSE	-
start	untyped	-		-
trace	logical	FALSE	TRUE, FALSE	-
x	logical	FALSE	TRUE, FALSE	-
y	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::LearnerClassif` -> `LearnerClassifLogReg`

**Methods****Public methods:**

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

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

*Usage:*

```
LearnerClassifLogReg$new()
```

**Method** `loglik()`: Extract the log-likelihood (e.g., via `stats::logLik()` from the fitted model.

*Usage:*

```
LearnerClassifLogReg$loglik()
```

**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

- Chapter in the [mlr3book](https://mlr3book.mlr-org.com/basics.html#learners): <https://mlr3book.mlr-org.com/basics.html#learners>
- Package [mlr3extralearners](#) for more learners.
- [Dictionary of Learners: 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

```
if (requireNamespace("stats", quietly = TRUE)) {
  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")
```

**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		$(-\infty, \infty)$
reitol	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\\$loglik\(\)](#)
- [LearnerClassifMultinom\\$clone\(\)](#)

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

*Usage:*

```
LearnerClassifMultinom$new()
```

**Method** `loglik()`: Extract the log-likelihood (e.g., via `stats::logLik()` from the fitted model.

*Usage:*

```
LearnerClassifMultinom$loglik()
```

**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

- Chapter in the [mlr3book](https://mlr3book.mlr-org.com/basics.html#learners): <https://mlr3book.mlr-org.com/basics.html#learners>
- Package [mlr3extralearners](#) for more learners.
- [Dictionary of Learners: 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

```
if (requireNamespace("nnet", quietly = TRUE)) {
  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")
```

## 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()`

**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

- Chapter in the [mlr3book](https://mlr3book.mlr-org.com/basics.html#learners): <https://mlr3book.mlr-org.com/basics.html#learners>
- Package [mlr3extralearners](#) for more learners.
- [Dictionary of Learners: 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

```
if (requireNamespace("e1071", quietly = TRUE)) {
  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")
```

## Meta Information

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

## Parameters

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

### Initial parameter values

- 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.

### See Also

- Chapter in the **mlr3book**: <https://mlr3book.mlr-org.com/basics.html#learners>
- Package **mlr3extralearners** for more learners.
- **Dictionary of Learners**: [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

```
if (requireNamespace("nnet", quietly = TRUE)) {
  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")
```

## 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()`

**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

- Chapter in the `mlr3book`: <https://mlr3book.mlr-org.com/basics.html#learners>
- Package `mlr3extralearners` for more learners.
- [Dictionary of Learners: 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

```
if (requireNamespace("MASS", quietly = TRUE)) {
  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 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: 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")
```

**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
alpha	numeric	0.5		$(-\infty, \infty)$
always.split.variables	untyped	-		-
class.weights	untyped			-
holdout	logical	FALSE	TRUE, FALSE	-
importance	character	-	none, impurity, impurity_corrected, permutation	-
keep.inbag	logical	FALSE	TRUE, FALSE	-
max.depth	integer	NULL		$[0, \infty)$
min.node.size	integer	NULL		$[1, \infty)$
min.prop	numeric	0.1		$(-\infty, \infty)$
minprop	numeric	0.1		$(-\infty, \infty)$
mtry	integer	-		$[1, \infty)$
mtry.ratio	numeric	-		$[0, 1]$
num.random.splits	integer	1		$[1, \infty)$
num.threads	integer	1		$[1, \infty)$
num.trees	integer	500		$[1, \infty)$
oob.error	logical	TRUE	TRUE, FALSE	-
regularization.factor	untyped	1		-
regularization.usedepth	logical	FALSE	TRUE, FALSE	-
replace	logical	TRUE	TRUE, FALSE	-
respect.unordered.factors	character	ignore	ignore, order, partition	-
sample.fraction	numeric	-		$[0, 1]$
save.memory	logical	FALSE	TRUE, FALSE	-
scale.permutation.importance	logical	FALSE	TRUE, FALSE	-
se.method	character	infjack	jack, infjack	-
seed	integer	NULL		$(-\infty, \infty)$
split.select.weights	untyped			-
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\\$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.

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/basics.html#learners): <https://mlr3book.mlr-org.com/basics.html#learners>
- Package [mlr3extralearners](#) for more learners.
- [Dictionary of Learners: 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

```
if (requireNamespace("ranger", quietly = TRUE)) {
  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")
```

## 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
cache.size	numeric	40		$(-\infty, \infty)$
class.weights	untyped			-
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()`

**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.

**See Also**

- Chapter in the [mlr3book](https://mlr3book.mlr-org.com/basics.html#learners): <https://mlr3book.mlr-org.com/basics.html#learners>
- Package [mlr3extralearners](#) for more learners.
- [Dictionary of Learners: 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, [mlr3tuningpaces](#) 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**

```
if (requireNamespace("e1071", quietly = TRUE)) {
  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](#).

If not specified otherwise, the evaluation metric is set to the default "logloss" for binary classification problems and set to "mlogloss" for multiclass problems. This was necessary to silence a deprecation warning.

Note that using the `watchlist` parameter directly will lead to problems when wrapping this [Learner](#) in a `mlr3pipelines` `GraphLearner` as the preprocessing steps will not be applied to the data in the `watchlist`.

### Initial parameter values

- 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.

### Early stopping

Early stopping can be used to find the optimal number of boosting rounds. The `early_stopping_set` parameter controls which set is used to monitor the performance. Set `early_stopping_set = "test"` to monitor the performance of the model on the test set while training. The test set for early stopping can be set with the "test" row role in the [mlr3::Task](#). Additionally, the range must be set in which the performance must increase with `early_stopping_rounds` and the maximum number of boosting rounds with `nrounds`. While resampling, the test set is automatically applied from the [mlr3::Resampling](#). Not that using the test set for early stopping can potentially bias the performance scores. See the section on early stopping in the examples.

### 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")
```

### 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	0.5		$(-\infty, \infty)$
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		$[0, 1]$
disable_default_eval_metric	logical	FALSE	TRUE, FALSE	-
early_stopping_rounds	integer	NULL		$[1, \infty)$
early_stopping_set	character	none	none, train, test	-
eta	numeric	0.3		$[0, 1]$
eval_metric	untyped	-		-
feature_selector	character	cyclic	cyclic, shuffle, random, greedy, thrifty	-
feval	untyped			-
gamma	numeric	0		$[0, \infty)$
grow_policy	character	depthwise	depthwise, lossguide	-
interaction_constraints	untyped	-		-
iterationrange	untyped	-		-
lambda	numeric	1		$[0, \infty)$
lambda_bias	numeric	0		$[0, \infty)$
max_bin	integer	256		$[2, \infty)$
max_delta_step	numeric	0		$[0, \infty)$
max_depth	integer	6		$[0, \infty)$
max_leaves	integer	0		$[0, \infty)$
maximize	logical	NULL	TRUE, FALSE	-
min_child_weight	numeric	1		$[0, \infty)$
missing	numeric	NA		$(-\infty, \infty)$
monotone_constraints	untyped	0		-
normalize_type	character	tree	tree, forest	-
nrounds	integer	-		$[1, \infty)$
nthread	integer	1		$[1, \infty)$
ntreelimit	integer	NULL		$[1, \infty)$
num_parallel_tree	integer	1		$[1, \infty)$
objective	untyped	binary:logistic		-
one_drop	logical	FALSE	TRUE, FALSE	-
outputmargin	logical	FALSE	TRUE, FALSE	-
predcontrib	logical	FALSE	TRUE, FALSE	-
predictor	character	cpu_predictor	cpu_predictor, gpu_predictor	-
predinteraction	logical	FALSE	TRUE, FALSE	-
predleaf	logical	FALSE	TRUE, FALSE	-
print_every_n	integer	1		$[1, \infty)$
process_type	character	default	default, update	-
rate_drop	numeric	0		$[0, 1]$
refresh_leaf	logical	TRUE	TRUE, FALSE	-
reshape	logical	FALSE	TRUE, FALSE	-

seed_per_iteration	logical	FALSE	TRUE, FALSE	-
sampling_method	character	uniform	uniform, gradient_based	-
sample_type	character	uniform	uniform, weighted	-
save_name	untyped			-
save_period	integer	NULL		$[0, \infty)$
scale_pos_weight	numeric	1		$(-\infty, \infty)$
skip_drop	numeric	0		$[0, 1]$
strict_shape	logical	FALSE	TRUE, FALSE	-
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	-		-
verbose	integer	1		$[0, 2]$
watchlist	untyped			-
xgb_model	untyped			-

## 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.

**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>.

**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/basics.html#learners>
- Package **mlr3extralearners** for more learners.
- **Dictionary of Learners**: [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**

```
if (requireNamespace("xgboost", quietly = TRUE)) {
  learner = mlr3::lrn("classif.xgboost")
  print(learner)

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

# Train learner with early stopping on spam data set
task = tsk("spam")

# Split task into training and test set
split = partition(task, ratio = 0.8)
task$set_row_roles(split$test, "test")
```

```
# Set early stopping parameter
learner = lrn("classif.xgboost",
  nrounds = 1000,
  early_stopping_rounds = 100,
  early_stopping_set = "test"
)

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

---

```
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")
```

## Meta Information

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

## Parameters

Id	Type	Default	Levels	Range
alignment	character	lambda	lambda, fraction	-
alpha	numeric	1		[0, 1]
big	numeric	9.9e+35		$(-\infty, \infty)$
devmax	numeric	0.999		[0, 1]
dfmax	integer	-		[0, $\infty$ )
eps	numeric	1e-06		[0, 1]
epsnr	numeric	1e-08		[0, 1]
exclude	integer	-		[1, $\infty$ )

exmx	numeric	250		$(-\infty, \infty)$
family	character	gaussian	gaussian, poisson	-
fdev	numeric	1e-05		[0, 1]
foldid	untyped			-
gamma	untyped	-		-
grouped	logical	TRUE	TRUE, FALSE	-
intercept	logical	TRUE	TRUE, FALSE	-
keep	logical	FALSE	TRUE, FALSE	-
lambda	untyped	-		-
lambda.min.ratio	numeric	-		[0, 1]
lower.limits	untyped	-		-
maxit	integer	100000		[1, $\infty$ )
mnlam	integer	5		[1, $\infty$ )
mxit	integer	100		[1, $\infty$ )
mxitnr	integer	25		[1, $\infty$ )
nfolds	integer	10		[3, $\infty$ )
nlambda	integer	100		[1, $\infty$ )
offset	untyped			-
parallel	logical	FALSE	TRUE, FALSE	-
penalty.factor	untyped	-		-
pmax	integer	-		[0, $\infty$ )
pmin	numeric	1e-09		[0, 1]
prec	numeric	1e-10		$(-\infty, \infty)$
predict.gamma	numeric	gamma.1se		$(-\infty, \infty)$
relax	logical	FALSE	TRUE, FALSE	-
s	numeric	lambda.1se		[0, $\infty$ )
standardize	logical	TRUE	TRUE, FALSE	-
standardize.response	logical	FALSE	TRUE, FALSE	-
thresh	numeric	1e-07		[0, $\infty$ )
trace.it	integer	0		[0, 1]
type.gaussian	character	-	covariance, naive	-
type.logistic	character	-	Newton, modified.Newton	-
type.measure	character	deviance	deviance, class, auc, mse, mae	-
type.multinomial	character	-	ungrouped, grouped	-
upper.limits	untyped	-		-

## Super classes

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

## Methods

### Public methods:

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

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

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

*Usage:*

```
LearnerRegrCVGlmnet$new()
```

**Method** `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.

**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.

## See Also

- Chapter in the [mlr3book](https://mlr3book.mlr-org.com/basics.html#learners): <https://mlr3book.mlr-org.com/basics.html#learners>
- Package [mlr3extralearners](#) for more learners.
- [Dictionary of Learners: 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, [mlr3tuningpaces](#) 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

```
if (requireNamespace("glmnet", quietly = TRUE)) {
  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".

## 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 lambdas would be more efficient, as is done by default in `glmnet::glmnet()`, tuning/selecting the parameter at prediction time (using parameter s) is currently not supported in **mlr3** (at least not in efficient manner). Tuning the s parameter is, therefore, currently discouraged.

When the data are i.i.d. and efficiency is key, we recommend using the respective auto-tuning 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 **Learner** can be instantiated via the dictionary `mlr_learners` or with the associated sugar function `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
alignment	character	lambda	lambda, fraction	-
alpha	numeric	1		[0, 1]
big	numeric	9.9e+35		$(-\infty, \infty)$
devmax	numeric	0.999		[0, 1]
dfmax	integer	-		[0, $\infty$ )
eps	numeric	1e-06		[0, 1]
epsnr	numeric	1e-08		[0, 1]
exact	logical	FALSE	TRUE, FALSE	-
exclude	integer	-		[1, $\infty$ )
exmx	numeric	250		$(-\infty, \infty)$
family	character	gaussian	gaussian, poisson	-
fdev	numeric	1e-05		[0, 1]
gamma	numeric	1		$(-\infty, \infty)$
grouped	logical	TRUE	TRUE, FALSE	-
intercept	logical	TRUE	TRUE, FALSE	-
keep	logical	FALSE	TRUE, FALSE	-
lambda	untyped	-		-
lambda.min.ratio	numeric	-		[0, 1]
lower.limits	untyped	-		-
maxit	integer	100000		[1, $\infty$ )
mnlam	integer	5		[1, $\infty$ )
mxit	integer	100		[1, $\infty$ )
mxitnr	integer	25		[1, $\infty$ )
newoffset	untyped	-		-
nlambda	integer	100		[1, $\infty$ )
offset	untyped	-		-
parallel	logical	FALSE	TRUE, FALSE	-
penalty.factor	untyped	-		-
pmax	integer	-		[0, $\infty$ )
pmin	numeric	1e-09		[0, 1]
prec	numeric	1e-10		$(-\infty, \infty)$
relax	logical	FALSE	TRUE, FALSE	-
s	numeric	0.01		[0, $\infty$ )
standardize	logical	TRUE	TRUE, FALSE	-
standardize.response	logical	FALSE	TRUE, FALSE	-
thresh	numeric	1e-07		[0, $\infty$ )
trace.it	integer	0		[0, 1]
type.gaussian	character	-	covariance, naive	-
type.logistic	character	-	Newton, modified.Newton	-
type.multinomial	character	-	ungrouped, grouped	-
upper.limits	untyped	-		-



**Super classes**

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

**Methods****Public methods:**

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

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

*Usage:*

`LearnerRegrGlmnet$new()`

**Method** `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.

**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.

**See Also**

- Chapter in the `mlr3book`: <https://mlr3book.mlr-org.com/basics.html#learners>
- Package `mlr3extralearners` for more learners.
- Dictionary of Learners: `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

```
if (requireNamespace("glmnet", quietly = TRUE)) {
  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**.

## Initial parameter values

- `store_model`:
  - See note.

## 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")
```

## 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, optimal
scale	logical	TRUE	TRUE, FALSE
ykernel	untyped		
store_model	logical	FALSE	TRUE, FALSE

**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()`.
- `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.ml-org.com/basics.html#learners>
- Package **mlr3extralearners** for more learners.
- **Dictionary of Learners**: [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

```
if (requireNamespace("kknn", quietly = TRUE)) {
  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")
```

## 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
<code>bias.correct</code>	logical	FALSE	TRUE, FALSE	-
<code>checkNames</code>	logical	TRUE	TRUE, FALSE	-
<code>coef.cov</code>	untyped			-
<code>coef.trend</code>	untyped			-
<code>coef.var</code>	untyped			-
<code>control</code>	untyped			-
<code>cov.compute</code>	logical	TRUE	TRUE, FALSE	-
<code>covtype</code>	character	<code>matern5_2</code>	<code>gauss</code> , <code>matern5_2</code> , <code>matern3_2</code> , <code>exp</code> , <code>powexp</code>	-
<code>estim.method</code>	character	MLE	MLE, LOO	-
<code>gr</code>	logical	TRUE	TRUE, FALSE	-
<code>iso</code>	logical	FALSE	TRUE, FALSE	-
<code>jitter</code>	numeric	0		$[0, \infty)$
<code>kernel</code>	untyped			-

knots	untyped			-
light.return	logical	FALSE	TRUE, FALSE	-
lower	untyped			-
multistart	integer	1		$(-\infty, \infty)$
noise.var	untyped			-
nugget	numeric	-		$(-\infty, \infty)$
nugget.estim	logical	FALSE	TRUE, FALSE	-
nugget.stability	numeric	0		$[0, \infty)$
optim.method	character	BFGS	BFGS, gen	-
parinit	untyped			-
penalty	untyped			-
scaling	logical	FALSE	TRUE, FALSE	-
se.compute	logical	TRUE	TRUE, FALSE	-
type	character	SK	SK, UK	-
upper	untyped			-

## 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.

**See Also**

- Chapter in the [mlr3book](https://mlr3book.mlr-org.com/basics.html#learners): <https://mlr3book.mlr-org.com/basics.html#learners>
- Package [mlr3extralearners](#) for more learners.
- [Dictionary of Learners: 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**

```
if (requireNamespace("DiceKriging", quietly = TRUE)) {
  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")
```

### 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	-
offset	logical	-	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	-

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

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

*Usage:*



```
LearnerRegrLM$new()
```

**Method** `loglik()`: Extract the log-likelihood (e.g., via `stats::logLik()`) from the fitted model.

*Usage:*

```
LearnerRegrLM$loglik()
```

**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

- Chapter in the [mlr3book](https://mlr3book.mlr-org.com/basics.html#learners): <https://mlr3book.mlr-org.com/basics.html#learners>
- Package [mlr3extralearners](#) for more learners.
- [Dictionary of Learners: 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

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

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

---

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 **mlr3keras**.

### Dictionary

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

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

### Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “numeric”, “factor”, “ordered”
- Required Packages: **mlr3**, **mlr3learners**, **nnet**

### Parameters

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

### Initial parameter values

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

### Super classes

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

### Methods

#### Public methods:

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

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

*Usage:*

```
LearnerRegrNnet$new()
```

**Method** `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/basics.html#learners>
- Package `mlr3extralearners` for more learners.
- [Dictionary of Learners: 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

```
if (requireNamespace("nnet", quietly = TRUE)) {
  learner = mlr3::lrn("regr.nnet")
  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")
```

## Meta Information

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

**Parameters**

Id	Type	Default	Levels	Range
alpha	numeric	0.5		$(-\infty, \infty)$
always.split.variables	untyped	-		-
holdout	logical	FALSE	TRUE, FALSE	-
importance	character	-	none, impurity, impurity_corrected, permutation	-
keep.inbag	logical	FALSE	TRUE, FALSE	-
max.depth	integer	NULL		$[0, \infty)$
min.node.size	integer	5		$[1, \infty)$
min.prop	numeric	0.1		$(-\infty, \infty)$
minprop	numeric	0.1		$(-\infty, \infty)$
mtry	integer	-		$[1, \infty)$
mtry.ratio	numeric	-		$[0, 1]$
num.random.splits	integer	1		$[1, \infty)$
num.threads	integer	1		$[1, \infty)$
num.trees	integer	500		$[1, \infty)$
oob.error	logical	TRUE	TRUE, FALSE	-
quantreg	logical	FALSE	TRUE, FALSE	-
regularization.factor	untyped	1		-
regularization.usedepth	logical	FALSE	TRUE, FALSE	-
replace	logical	TRUE	TRUE, FALSE	-
respect.unordered.factors	character	ignore	ignore, order, partition	-
sample.fraction	numeric	-		$[0, 1]$
save.memory	logical	FALSE	TRUE, FALSE	-
scale.permutation.importance	logical	FALSE	TRUE, FALSE	-
se.method	character	infjack	jack, infjack	-
seed	integer	NULL		$(-\infty, \infty)$
split.select.weights	untyped			-
splitrule	character	variance	variance, extratrees, maxstat	-
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  $mtry = \max(\text{ceiling}(mtry.ratio * n\_features), 1)$ . Note that mtry and mtry.ratio are mutually exclusive.

**Initial parameter values**

- 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.

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/basics.html#learners): <https://mlr3book.mlr-org.com/basics.html#learners>
- Package [mlr3extralearners](#) for more learners.
- [Dictionary of Learners: 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, [mlr3tuningpaces](#) 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**

```
if (requireNamespace("ranger", quietly = TRUE)) {
  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")
```

**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()`

**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.

## See Also

- Chapter in the [mlr3book](https://mlr3book.mlr-org.com/basics.html#learners): <https://mlr3book.mlr-org.com/basics.html#learners>
- Package [mlr3extralearners](#) for more learners.
- [Dictionary of Learners: 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

```
if (requireNamespace("e1071", quietly = TRUE)) {
  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](#).

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 `watchlist` parameter directly will lead to problems when wrapping this [Learner](#) in a [mlr3pipelines](#) `GraphLearner` as the preprocessing steps will not be applied to the data in the `watchlist`.

## 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")
```

## 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	0.5		$(-\infty, \infty)$
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		$[0, 1]$
disable_default_eval_metric	logical	FALSE	TRUE, FALSE	-
early_stopping_rounds	integer	NULL		$[1, \infty)$
early_stopping_set	character	none	none, train, test	-
eta	numeric	0.3		$[0, 1]$
eval_metric	untyped	rmse		-
feature_selector	character	cyclic	cyclic, shuffle, random, greedy, thrifty	-
feval	untyped			-
gamma	numeric	0		$[0, \infty)$
grow_policy	character	depthwise	depthwise, lossguide	-
interaction_constraints	untyped	-		-
iterationrange	untyped	-		-
lambda	numeric	1		$[0, \infty)$
lambda_bias	numeric	0		$[0, \infty)$
max_bin	integer	256		$[2, \infty)$
max_delta_step	numeric	0		$[0, \infty)$
max_depth	integer	6		$[0, \infty)$
max_leaves	integer	0		$[0, \infty)$
maximize	logical	NULL	TRUE, FALSE	-
min_child_weight	numeric	1		$[0, \infty)$
missing	numeric	NA		$(-\infty, \infty)$
monotone_constraints	untyped	0		-
normalize_type	character	tree	tree, forest	-

nrounds	integer	-		[1, ∞)
nthread	integer	1		[1, ∞)
ntreelimit	integer	NULL		[1, ∞)
num_parallel_tree	integer	1		[1, ∞)
objective	untyped	reg:squarederror		-
one_drop	logical	FALSE	TRUE, FALSE	-
outputmargin	logical	FALSE	TRUE, FALSE	-
predcontrib	logical	FALSE	TRUE, FALSE	-
predictor	character	cpu_predictor	cpu_predictor, gpu_predictor	-
predinteraction	logical	FALSE	TRUE, FALSE	-
predleaf	logical	FALSE	TRUE, FALSE	-
print_every_n	integer	1		[1, ∞)
process_type	character	default	default, update	-
rate_drop	numeric	0		[0, 1]
refresh_leaf	logical	TRUE	TRUE, FALSE	-
reshape	logical	FALSE	TRUE, FALSE	-
sampling_method	character	uniform	uniform, gradient_based	-
sample_type	character	uniform	uniform, weighted	-
save_name	untyped			-
save_period	integer	NULL		[0, ∞)
scale_pos_weight	numeric	1		(-∞, ∞)
seed_per_iteration	logical	FALSE	TRUE, FALSE	-
skip_drop	numeric	0		[0, 1]
strict_shape	logical	FALSE	TRUE, FALSE	-
subsample	numeric	1		[0, 1]
top_k	integer	0		[0, ∞)
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	-		-
verbose	integer	1		[0, 2]
watchlist	untyped			-
xgb_model	untyped			-

## Early stopping

Early stopping can be used to find the optimal number of boosting rounds. The `early_stopping_set` parameter controls which set is used to monitor the performance. Set `early_stopping_set = "test"` to monitor the performance of the model on the test set while training. The test set for early stopping can be set with the "test" row role in the `mlr3::Task`. Additionally, the range must be set in which the performance must increase with `early_stopping_rounds` and the maximum number of boosting rounds with `nrounds`. While resampling, the test set is automatically applied from the `mlr3::Resampling`. Not that using the test set for early stopping can potentially bias the performance scores. See the section on early stopping in the examples.

**Initial parameter values**

- 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.

**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>.

## 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/basics.html#learners>
- Package **mlr3extralearners** for more learners.
- **Dictionary of Learners**: [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, **mlr3tuningpaces** 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

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

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

# Train learner with early stopping on spam data set
task = tsk("mtcars")

# Split task into training and test set
split = partition(task, ratio = 0.8)
task$set_row_roles(split$test, "test")

# Set early stopping parameter
learner = lrn("regr.xgboost",
  nrounds = 1000,
  early_stopping_rounds = 100,
  early_stopping_set = "test"
)
```

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

# Index

## \* Learner

- mlr\_learners\_classif.cv\_glmnet, 3
  - mlr\_learners\_classif.glmnet, 6
  - mlr\_learners\_classif.kknn, 9
  - mlr\_learners\_classif.lda, 12
  - mlr\_learners\_classif.log\_reg, 14
  - mlr\_learners\_classif.multinom, 16
  - mlr\_learners\_classif.naive\_bayes, 19
  - mlr\_learners\_classif.nnet, 21
  - mlr\_learners\_classif.qda, 23
  - mlr\_learners\_classif.ranger, 26
  - mlr\_learners\_classif.svm, 29
  - mlr\_learners\_classif.xgboost, 31
  - mlr\_learners\_regr.cv\_glmnet, 36
  - mlr\_learners\_regr.glmnet, 39
  - mlr\_learners\_regr.kknn, 42
  - mlr\_learners\_regr.km, 45
  - mlr\_learners\_regr.lm, 47
  - mlr\_learners\_regr.nnet, 50
  - mlr\_learners\_regr.ranger, 52
  - mlr\_learners\_regr.svm, 55
  - mlr\_learners\_regr.xgboost, 57
- contr.poly(), 15, 48
- contr.treatment(), 15, 48
- DiceKriging::km(), 45
- Dictionary, 5, 9, 11, 13, 16, 18, 20, 22, 25, 28, 31, 35, 38, 41, 44, 47, 49, 51, 55, 57, 61
- dictionary, 3, 7, 10, 12, 14, 17, 19, 21, 23, 26, 29, 32, 36, 39, 42, 45, 47, 50, 52, 55, 58
- e1071::naiveBayes(), 19
- e1071::svm(), 29, 55
- glmnet::cv.glmnet(), 3, 6, 36, 39
- glmnet::glmnet(), 6, 39
- glmnet::predict.glmnet(), 5, 8, 38, 41
- kknn::kknn(), 9, 11, 42, 43
- Learner, 3, 7, 10, 12, 14, 17, 19, 21, 23, 26, 29, 31, 32, 36, 39, 42, 45, 47, 50, 52, 55, 57, 58
- LearnerClassifCVGlmnet  
(mlr\_learners\_classif.cv\_glmnet), 3
- LearnerClassifGlmnet  
(mlr\_learners\_classif.glmnet), 6
- LearnerClassifKKNN  
(mlr\_learners\_classif.kknn), 9
- LearnerClassifLDA  
(mlr\_learners\_classif.lda), 12
- LearnerClassifLogReg  
(mlr\_learners\_classif.log\_reg), 14
- LearnerClassifMultinom  
(mlr\_learners\_classif.multinom), 16
- LearnerClassifNaiveBayes  
(mlr\_learners\_classif.naive\_bayes), 19
- LearnerClassifNnet  
(mlr\_learners\_classif.nnet), 21
- LearnerClassifQDA  
(mlr\_learners\_classif.qda), 23
- LearnerClassifRanger  
(mlr\_learners\_classif.ranger), 26
- LearnerClassifSVM  
(mlr\_learners\_classif.svm), 29
- LearnerClassifXgboost  
(mlr\_learners\_classif.xgboost), 31
- LearnerRegrCVGlmnet  
(mlr\_learners\_regr.cv\_glmnet),

- 36
- LearnerRegrGlmnet  
(mlr\_learners\_regr.glmnet), 39
- LearnerRegrKknn  
(mlr\_learners\_regr.kknn), 42
- LearnerRegrKM (mlr\_learners\_regr.km), 45
- LearnerRegrLM (mlr\_learners\_regr.lm), 47
- LearnerRegrNnet  
(mlr\_learners\_regr.nnet), 50
- LearnerRegrRanger  
(mlr\_learners\_regr.ranger), 52
- LearnerRegrSVM (mlr\_learners\_regr.svm), 55
- LearnerRegrXgboost  
(mlr\_learners\_regr.xgboost), 57
- Learners, 5, 6, 9, 11, 13, 16, 18, 20, 22, 25, 28, 31, 35, 38, 41, 44, 47, 49, 51, 55, 57, 61
- lrn(), 3, 7, 10, 12, 14, 17, 19, 21, 23, 26, 29, 32, 36, 39, 42, 45, 47, 50, 52, 55, 58
- MASS::lda(), 12
- MASS::qda(), 23
- mlr3::Learner, 5, 8, 10, 13, 15, 17, 19, 22, 25, 27, 30, 34, 37, 41, 43, 46, 48, 51, 54, 56, 60
- mlr3::LearnerClassif, 5, 8, 10, 13, 15, 17, 19, 22, 25, 27, 30, 34
- mlr3::LearnerRegr, 37, 41, 43, 46, 48, 51, 54, 56, 60
- mlr3::Resampling, 32, 59
- mlr3::Task, 32, 59
- mlr3learners (mlr3learners-package), 2
- mlr3learners-package, 2
- mlr\_learners, 3, 5, 7, 9–14, 16–23, 25, 26, 28, 29, 31, 32, 35, 36, 38, 39, 41, 42, 44, 45, 47, 49–52, 55, 57, 58, 61
- mlr\_learners\_classif.cv\_glmnet, 3, 9, 11, 13, 16, 18, 20, 23, 26, 29, 31, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.cv\_glmnet(), 6, 39
- mlr\_learners\_classif.glmnet, 6, 6, 11, 13, 16, 18, 20, 23, 26, 29, 31, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.kknn, 6, 9, 9, 13, 16, 18, 20, 23, 26, 29, 31, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.lda, 6, 9, 11, 12, 16, 18, 20, 23, 26, 29, 31, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.log\_reg, 6, 9, 11, 13, 14, 18, 20, 23, 26, 29, 31, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.multinom, 6, 9, 11, 13, 16, 16, 20, 23, 26, 29, 31, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.naive\_bayes, 6, 9, 11, 13, 16, 18, 19, 23, 26, 29, 31, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.nnet, 6, 9, 11, 13, 16, 18, 20, 21, 26, 29, 31, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.qda, 6, 9, 11, 13, 16, 18, 20, 23, 23, 29, 31, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.ranger, 6, 9, 11, 13, 16, 18, 20, 23, 26, 26, 31, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.svm, 6, 9, 11, 13, 16, 18, 20, 23, 26, 29, 29, 35, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_classif.xgboost, 6, 9, 11, 13, 16, 18, 20, 23, 26, 29, 31, 31, 38, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_regr.cv\_glmnet, 6, 9, 11, 13, 16, 18, 20, 23, 26, 29, 31, 35, 36, 42, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_regr.cv\_glmnet(), 6, 39
- mlr\_learners\_regr.glmnet, 6, 9, 11, 13, 16, 18, 20, 23, 26, 29, 31, 35, 38, 39, 44, 47, 49, 52, 55, 57, 61
- mlr\_learners\_regr.kknn, 6, 9, 11, 13, 16, 18, 20, 23, 26, 29, 31, 35, 38, 42, 42, 47, 49, 52, 55, 57, 61
- mlr\_learners\_regr.km, 6, 9, 11, 13, 16, 18, 20, 23, 26, 29, 31, 35, 38, 42, 44, 45, 49, 52, 55, 57, 61
- mlr\_learners\_regr.lm, 6, 9, 11, 13, 16, 18, 20, 23, 26, 29, 31, 35, 38, 42, 44, 47, 47, 52, 55, 57, 61
- mlr\_learners\_regr.nnet, 6, 9, 11, 13, 16, 18, 20, 23, 26, 29, 31, 35, 38, 42, 44, 47, 49, 50, 55, 57, 61
- mlr\_learners\_regr.ranger, 6, 9, 11, 13, 16, 18, 20, 23, 26, 29, 31, 35, 38, 42, 44, 47, 49, 52, 52, 57, 61
- mlr\_learners\_regr.svm, 6, 9, 11, 13, 16, 18,



20, 23, 26, 29, 31, 35, 38, 42, 44, 47,  
49, 52, 55, 55, 61

`mlr_learners_regr.xgboost`, 6, 9, 11, 13,  
16, 18, 20, 23, 26, 29, 31, 35, 38, 42,  
44, 47, 49, 52, 55, 57, 57

`nnet::multinom()`, 16

`nnet::nnet.formula()`, 21, 50

R6, 5, 8, 10, 13, 15, 18, 19, 22, 25, 28, 30, 34,  
38, 41, 43, 46, 48, 51, 54, 56, 60

`ranger::ranger()`, 26, 52

`stats::glm()`, 5, 8, 14

`stats::lm()`, 47

`stats::logLik()`, 15, 18, 49

`xgboost::xgb.importance()`, 34, 60

`xgboost::xgb.train()`, 31, 57