Recommended Learners for 'mlr3'

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 (>= 0.20.0), R (>= 3.1.0)
Imports
checkmate, data.table, mlr3misc (>= 0.9.4), paradox (>= 1.0.0), 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
7.3.1
Collate
'aaa.R' 'LearnerClassifCVGlmnet.R' 'LearnerClassifGlmnet.R'
' LearnerClassifKNN.R' 'LearnerClassifLDA.R'
' LearnerClassifLogReg.R' 'LearnerClassifMultinom.R'
' LearnerClassifNaiveBayes.R' 'LearnerClassifNnet.R'
' LearnerClassifQDA.R' 'LearnerClassifRanger.R'
' LearnerClassifSVM.R' 'LearnerClassifXgboost.R'
' LearnerRegrCVGlmnet.R' 'LearnerRegrGlmnet.R'
' LearnerRegrKNN.R' 'LearnerRegrKM.R' 'LearnerRegrLM.R'
' LearnerRegrNnet.R' 'LearnerRegrRanger.R' 'LearnerRegrSVM.R'
mlr3learners-package

mlr3learners: Recommended Learners for 'mlr3'

Description

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

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

Authors:

- Michel Lang <michellang@gmail.com> (ORCID)
- Quay Au <quayau@gmail.com> (ORCID)
- Stefan Coors <mail@stefancoors.de> (ORCID)
- Patrick Schratz <patrick.schratz@gmail.com> (ORCID)

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 \texttt{glmnet::cv.glmnet()} from package \texttt{glmnet}.

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

Dictionary

This \texttt{mlr3::Learner} can be instantiated via the dictionary \texttt{mlr3::mlr_learners} or with the associated sugar function \texttt{mlr3::lrn()}:

\begin{verbatim}
mlr_learners$get("classif.cv_glmnet")
lrn("classif.cv_glmnet")
\end{verbatim}

Meta Information

- Task type: "classif"
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: \texttt{mlr3, mlr3learners, glmnet}
### Parameters

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>alignment</td>
<td>character</td>
<td>lambda</td>
<td>lambda, fraction</td>
<td>-</td>
</tr>
<tr>
<td>alpha</td>
<td>numeric</td>
<td>1</td>
<td>-</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>big</td>
<td>numeric</td>
<td>9.9e+35</td>
<td>-</td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>devmax</td>
<td>numeric</td>
<td>0.999</td>
<td>-</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>dfmax</td>
<td>integer</td>
<td>-</td>
<td>-</td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>epsnr</td>
<td>numeric</td>
<td>1e-08</td>
<td>-</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>eps</td>
<td>numeric</td>
<td>1e-06</td>
<td>-</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>exclude</td>
<td>integer</td>
<td>-</td>
<td>-</td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>exmx</td>
<td>numeric</td>
<td>250</td>
<td>-</td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>fdev</td>
<td>numeric</td>
<td>1e-05</td>
<td>-</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>foldid</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>gamma</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>grouped</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>intercept</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>keep</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>lambdaminratio</td>
<td>numeric</td>
<td>-</td>
<td>-</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>lambda</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>lowerlimits</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>maxit</td>
<td>integer</td>
<td>100000</td>
<td>-</td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>mmult</td>
<td>integer</td>
<td>5</td>
<td>-</td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>mxtitr</td>
<td>integer</td>
<td>25</td>
<td>-</td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>mxit</td>
<td>integer</td>
<td>100</td>
<td>-</td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>nfolds</td>
<td>integer</td>
<td>10</td>
<td>-</td>
<td>[3, ∞)</td>
</tr>
<tr>
<td>nlambda</td>
<td>integer</td>
<td>100</td>
<td>-</td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>offset</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>parallel</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>penalty.factor</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>pmax</td>
<td>integer</td>
<td>-</td>
<td>-</td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>pmin</td>
<td>numeric</td>
<td>1e-09</td>
<td>-</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>prec</td>
<td>numeric</td>
<td>1e-10</td>
<td>-</td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>predict.gamma</td>
<td>numeric</td>
<td>gamma.1se</td>
<td>-</td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>relax</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>s</td>
<td>numeric</td>
<td>lambda.1se</td>
<td>-</td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>standardize</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>standardizeresponse</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>thresh</td>
<td>numeric</td>
<td>1e-07</td>
<td>-</td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>trace.it</td>
<td>integer</td>
<td>0</td>
<td>-</td>
<td>[0, 1]</td>
</tr>
<tr>
<td>type.gaussian</td>
<td>character</td>
<td>-</td>
<td>covariance, naive</td>
<td>-</td>
</tr>
<tr>
<td>type.logistic</td>
<td>character</td>
<td>-</td>
<td>Newton, modified.Newton</td>
<td>-</td>
</tr>
<tr>
<td>type.measure</td>
<td>character</td>
<td>deviance</td>
<td>deviance, class, auc, mse, mae</td>
<td>-</td>
</tr>
<tr>
<td>type.multinomial</td>
<td>character</td>
<td>-</td>
<td>ungrouped, grouped</td>
<td>-</td>
</tr>
<tr>
<td>upperlimits</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
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:
depth Whether to make a deep clone.

References


See Also

• Package mlr3extralearners for more learners.
• Dictionary of Learners: mlr3::mlr_learners
• as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
• mlr3pipelines to combine learners with pre- and postprocessing steps.
• Extension packages for additional task types:
  – mlr3proba for probabilistic supervised regression and survival analysis.
  – mlr3cluster for unsupervised clustering.
• mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.


Examples

```r
if (requireNamespace("glmnet", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.cv_glmnet")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

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

mlr_learners_classif.glmnet

GLM with Elastic Net Regularization Classification Learner
**Description**

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

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

```r
mlr3::lrn("classif.glmnet")
```

**Meta Information**

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

**Parameters**

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>numeric</td>
<td>1</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>big</td>
<td>numeric</td>
<td>9.9e+35</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>devmax</td>
<td>numeric</td>
<td>0.999</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>dfmax</td>
<td>integer</td>
<td>-</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>eps</td>
<td>numeric</td>
<td>1e-06</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>epsnr</td>
<td>numeric</td>
<td>1e-08</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>exact</td>
<td>logical</td>
<td>FALSE</td>
<td></td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>exclude</td>
<td>integer</td>
<td>-</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>exmx</td>
<td>numeric</td>
<td>250</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>fdev</td>
<td>numeric</td>
<td>1e-05</td>
<td></td>
<td>[0, 1]</td>
</tr>
</tbody>
</table>
### 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
NULL
-

### 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
-
-

---

**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 \texttt{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

See Also
• Chapter in the mlr3book: \url{https://mlr3book.mlr-org.com/chapters/chapter2/data_and_basic_modeling.html#sec-learners}
• Package mlr3extralearners for more learners.
• Dictionary of Learners: mlr3::mlr_learners
• \texttt{as.data.table(mlr_learners)} for a table of available Learners in the running session (depending on the loaded packages).
• \texttt{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.

Examples

```r
if (requireNamespace("glmnet", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.glmnet")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

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

mlr_learners_classif.kknn

_k-Nearest-Neighbor Classification Learner_

Description

_k-Nearest-Neighbor classification. Calls `kknn::kknn()` from package `kknn`._

Initial parameter values

- **store_model:**
  - See note.

Dictionary

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

```r
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>integer</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>distance</td>
<td>numeric</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>kernel</td>
<td>character</td>
<td>optimal</td>
<td>rectangular, triangular, epanechnikov, biweight, triweight, cos, inv, gaussian, rank, optimal</td>
</tr>
<tr>
<td>scale</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>ykernel</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
</tr>
<tr>
<td>store_model</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
</tr>
</tbody>
</table>

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().
mlr_learners_classif.kknn

- **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**


**See Also**

- Package mlr3extrainlearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- mlr3pipelines to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - mlr3proba for probabilistic supervised regression and survival analysis.
  - mlr3cluster for unsupervised clustering.
- mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.


**Examples**

```r
if (requireNamespace("kknn", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.kknn")
  print(learner)

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

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

# print the model
print(learner$model)

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

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

# Score the predictions
predictions$score()

mlr_learners_classif.lda

Linear Discriminant Analysis Classification Learner

Description

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

Details

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

Dictionary

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

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

Meta Information

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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimen</td>
<td>untyped</td>
<td>-</td>
<td>moment, mle, mve, t</td>
<td>-</td>
</tr>
<tr>
<td>method</td>
<td>character</td>
<td>moment</td>
<td>moment, mle, mve, t</td>
<td>-</td>
</tr>
<tr>
<td>nu</td>
<td>integer</td>
<td>-</td>
<td>(−∞, ∞)</td>
<td></td>
</tr>
<tr>
<td>predict.method</td>
<td>character</td>
<td>plug-in</td>
<td>plug-in, predictive, debiased</td>
<td>-</td>
</tr>
<tr>
<td>predict.prior</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>prior</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>tol</td>
<td>numeric</td>
<td>-</td>
<td>(−∞, ∞)</td>
<td></td>
</tr>
</tbody>
</table>

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


See Also

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
• **mlr3pipelines** to combine learners with pre- and postprocessing steps.
• Extension packages for additional task types:
  – **mlr3proba** for probabilistic supervised regression and survival analysis.
  – **mlr3cluster** for unsupervised clustering.
• **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.


**Examples**

```r
if (requireNamespace("MASS", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.lda")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

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

---

**mlr_learners_classif.log_reg**

*Logistic Regression Classification Learner*

**Description**

Classification via logistic regression. Calls `stats::glm()` with family set to "binomial".
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 mlr3::Learner can be instantiated via the dictionary mlr3::mlr_learners or with the associated sugar function mlr3::lrn():

```r
mlr_learners$get("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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>dispersion</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>epsilon</td>
<td>numeric</td>
<td>1e-08</td>
<td>(−∞, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>etastart</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>maxit</td>
<td>numeric</td>
<td>25</td>
<td>(−∞, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>model</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>mustart</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>offset</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>singular.ok</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>start</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>trace</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>x</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>y</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>
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:

depth Whether to make a deep clone.

See Also

- Package `mlr3extralearners` for more learners.
- Dictionary of Learners: `mlr3::mlr_learners`
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- `mlr3pipelines` to combine learners with pre- and postprocessing steps.
• Extension packages for additional task types:
  – **mlr3proba** for probabilistic supervised regression and survival analysis.
  – **mlr3cluster** for unsupervised clustering.
• **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.


**Examples**

```r
if (requireNamespace("stats", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.log_reg")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

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

---

**mlr_learners_classif.multinom**

*Multinomial log-linear learner via neural networks*

**Description**

Dictionary

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

```r
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hess</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>abstol</td>
<td>numeric</td>
<td>1e-04</td>
<td>(-∞, ∞)</td>
<td></td>
</tr>
<tr>
<td>censored</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>decay</td>
<td>numeric</td>
<td>0</td>
<td>(-∞, ∞)</td>
<td></td>
</tr>
<tr>
<td>entropy</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>mask</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>maxit</td>
<td>integer</td>
<td>100</td>
<td>[1, ∞)</td>
<td></td>
</tr>
<tr>
<td>MaxNWts</td>
<td>integer</td>
<td>1000</td>
<td>[1, ∞)</td>
<td></td>
</tr>
<tr>
<td>model</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>linout</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>rang</td>
<td>numeric</td>
<td>0.7</td>
<td>(-∞, ∞)</td>
<td></td>
</tr>
<tr>
<td>reltol</td>
<td>numeric</td>
<td>1e-08</td>
<td>(-∞, ∞)</td>
<td></td>
</tr>
<tr>
<td>size</td>
<td>integer</td>
<td>-</td>
<td>[1, ∞)</td>
<td></td>
</tr>
<tr>
<td>skip</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>softmax</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>summ</td>
<td>character</td>
<td>0</td>
<td>0, 1, 2, 3</td>
<td></td>
</tr>
<tr>
<td>trace</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>Wts</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- mlr3pipelines to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - mlr3proba for probabilistic supervised regression and survival analysis.
  - mlr3cluster for unsupervised clustering.
- mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.

Examples

```r
if (requireNamespace("nnet", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.multinom")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

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

---

**mlr_learners_classif.naive_bayes**

*Naive Bayes Classification Learner*

**Description**

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

**Dictionary**

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

```r
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>eps</td>
<td>numeric</td>
<td>0</td>
<td>$(-\infty, \infty)$</td>
</tr>
<tr>
<td>laplace</td>
<td>numeric</td>
<td>0</td>
<td>$[0, \infty)$</td>
</tr>
<tr>
<td>threshold</td>
<td>numeric</td>
<td>0.001</td>
<td>$(-\infty, \infty)$</td>
</tr>
</tbody>
</table>

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

- Package `mlr3extralearners` for more learners.
- Dictionary of Learners: `mlr3::mlr_learners`
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- `mlr3pipelines` to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - `mlr3proba` for probabilistic supervised regression and survival analysis.
  - `mlr3cluster` for unsupervised clustering.
• **mlr3tuning** for tuning of hyperparameters, **mlr3tuningspaces** for established default tuning spaces.


**Examples**

```r
if (requireNamespace("e1071", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.naive_bayes")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

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

**mlr_learners_classif.nnet**

*Classification Neural Network Learner*

**Description**

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

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

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

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

Meta Information

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

Parameters

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hess</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>MaxNWts</td>
<td>integer</td>
<td>1000</td>
<td>[1, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>Wts</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>abstol</td>
<td>numeric</td>
<td>1e-04</td>
<td>(−∞, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>censored</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>contrasts</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>decay</td>
<td>numeric</td>
<td>0</td>
<td>(−∞, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>mask</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>maxit</td>
<td>integer</td>
<td>100</td>
<td>[1, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>na.action</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>rang</td>
<td>numeric</td>
<td>0.7</td>
<td>(−∞, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>reltol</td>
<td>numeric</td>
<td>1e-08</td>
<td>(−∞, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>size</td>
<td>integer</td>
<td>3</td>
<td>[0, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>skip</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>subset</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>trace</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>formula</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Initial parameter values

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

Custom mlr3 parameters

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

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

Methods

**Public methods:**

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

**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


See Also

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- mlr3pipelines to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - mlr3proba for probabilistic supervised regression and survival analysis.
  - mlr3cluster for unsupervised clustering.
- mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.

Examples

```r
if (requireNamespace("nnet", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.nnet")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

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

---

**mlr_learners_classif.qda**

*Quadratic Discriminant Analysis Classification Learner*

**Description**

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

**Details**

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

**Dictionary**

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

```r
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>character</td>
<td>moment</td>
<td>moment, mle, mve, t</td>
<td>-</td>
</tr>
<tr>
<td>nu</td>
<td>integer</td>
<td>-</td>
<td>plug-in, predictive, debiased</td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>predict.method</td>
<td>character</td>
<td>plug-in</td>
<td>plug-in, predictive, debiased</td>
<td>-</td>
</tr>
<tr>
<td>predict.prior</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>prior</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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

See Also

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- \texttt{as.data.table(mlr_learners)} for a table of available Learners in the running session (depending on the loaded packages).
- \texttt{mlr3pipelines} to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - \texttt{mlr3proba} for probabilistic supervised regression and survival analysis.
  - \texttt{mlr3cluster} for unsupervised clustering.
- \texttt{mlr3tuning} for tuning of hyperparameters, \texttt{mlr3tuningspaces} for established default tuning spaces.


Examples

```r
if (requireNamespace("MASS", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.qda")
  print(learner)
  
  # Define a Task
  task = tsk("sonar")
  
  # Create train and test set
  ids = partition(task)
  
  # Create the learner on the training ids
  learner$train(task, row_ids = ids$train)
  
  # Print the model
  print(learner$model)
  
  # Importance method
  if("importance" %in% learner$properties) print(learner$importance)
  
  # Make predictions for the test rows
  predictions = learner$predict(task, row_ids = ids$test)
  
  # Score the predictions
  predictions$score()
}
```
mlr_learners_classif.ranger

Ranger Classification Learner

Description

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

Custom mlr3 parameters

- `mtry`:
  - This hyperparameter can alternatively be set via our hyperparameter `mtry.ratio` as \( mtry = \max(\text{ceiling}(mtry.ratio \times n\text{-features}), 1) \). 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 `mlr3::Learner` can be instantiated via the dictionary `mlr3::mlr_learners` or with the associated sugar function `mlr3::lrn()`:

```r
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>numeric</td>
<td>0.5</td>
<td></td>
<td>((-\infty, \infty))</td>
</tr>
<tr>
<td>always.split.variables</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>class.weights</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>holdout</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>importance</td>
<td>character</td>
<td>-</td>
<td>none, impurity, impurity_corrected, permutation</td>
<td>-</td>
</tr>
</tbody>
</table>
### 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:
deepp Whether to make a deep clone.

References


See Also

• Package mlr3extralearners for more learners.
• Dictionary of Learners: mlr3::mlr_learners
• as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
• mlr3pipelines to combine learners with pre- and postprocessing steps.
• Extension packages for additional task types:
  – mlr3proba for probabilistic supervised regression and survival analysis.
  – mlr3cluster for unsupervised clustering.
• mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.

Examples

```r
if (requireNamespace("ranger", quietly = TRUE)) {
# Define the Learner and set parameter values
learner = lrn("classif.ranger")
print(learner)

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

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

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

# print the model
print(learner$model)

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

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

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

---

**mlr_learners_classif.svm**

Support Vector Machine

**Description**

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

**Dictionary**

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

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

```r
lrn("classif.svm")
```

**Meta Information**

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”
- Required Packages: `mlr3`, `mlr3learners`, `e1071`
### Parameters

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

### 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

See Also

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- mlr3pipelines to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - mlr3proba for probabilistic supervised regression and survival analysis.
  - mlr3cluster for unsupervised clustering.
- mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.


Examples

if (requireNamespace("e1071", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.svm")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

  # Score the predictions
  predictions$score()
}
Extreme Gradient Boosting Classification Learner

Description

eXtreme Gradient Boosting classification. Calls \texttt{xgboost::xgb.train()} from package \texttt{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 \texttt{mlr3::Learner} in a \texttt{mlr3pipelines GraphLearner} as the preprocessing steps will not be applied to the data in the watchlist. See the section \textit{Early Stopping and Validation} on how to do this.

Initial parameter values

- \textit{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. \texttt{nrounds} needs to be tuned by the user.

- \textit{nthread}:
  - Actual value: Undefined, triggering auto-detection of the number of CPUs.
  - Adjusted value: 1.
  - Reason for change: Conflicting with parallelization via \texttt{future}.

- \textit{verbose}:
  - Actual default: 1.
  - Adjusted default: 0.
  - Reason for change: Reduce verbosity.

Early Stopping and Validation

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

Dictionary

This \texttt{mlr3::Learner} can be instantiated via the \texttt{dictionary mlr3::mlr_learners} or with the associated sugar function \texttt{mlr3::lrn()}:

\begin{verbatim}
mlr_learners$gget("classif.xgboost")
lrn("classif.xgboost")
\end{verbatim}
**Meta Information**

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

**Parameters**

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>numeric</td>
<td>0</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>approxcontrib</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td></td>
</tr>
<tr>
<td>base_score</td>
<td>numeric</td>
<td>0.5</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>booster</td>
<td>character</td>
<td>gtree</td>
<td>gtree, glinear, dart</td>
<td></td>
</tr>
<tr>
<td>callbacks</td>
<td>untyped</td>
<td>list()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>colsample_bylevel</td>
<td>numeric</td>
<td>1</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>colsample_bybnode</td>
<td>numeric</td>
<td>1</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>colsample_bytree</td>
<td>numeric</td>
<td>1</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>device</td>
<td>untyped</td>
<td>&quot;cpu&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>disable_default_eval_metric</td>
<td>logical</td>
<td>TRUE, FALSE</td>
<td>TRUE, FALSE</td>
<td></td>
</tr>
<tr>
<td>early_stopping_rounds</td>
<td>integer</td>
<td>NULL</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>eta</td>
<td>numeric</td>
<td>0.3</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>eval_metric</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>feature_selector</td>
<td>character</td>
<td>cyclic</td>
<td>cyclic, shuffle, random, greedy, thrifty</td>
<td></td>
</tr>
<tr>
<td>feval</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gamma</td>
<td>numeric</td>
<td>0</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>grow_policy</td>
<td>character</td>
<td>depthwise</td>
<td>depthwise, lossguide</td>
<td></td>
</tr>
<tr>
<td>interaction_constraints</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iterationrange</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lambda</td>
<td>numeric</td>
<td>1</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>lambda_bias</td>
<td>numeric</td>
<td>0</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>max_bin</td>
<td>integer</td>
<td>256</td>
<td></td>
<td>[2, ∞)</td>
</tr>
<tr>
<td>max_delta_step</td>
<td>numeric</td>
<td>0</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>max_depth</td>
<td>integer</td>
<td>6</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>max_leaves</td>
<td>integer</td>
<td>0</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>maximize</td>
<td>logical</td>
<td>NULL</td>
<td>TRUE, FALSE</td>
<td></td>
</tr>
<tr>
<td>min_child_weight</td>
<td>numeric</td>
<td>1</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>missing</td>
<td>numeric</td>
<td>NA</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>monotone_constraints</td>
<td>untyped</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nrounds</td>
<td>integer</td>
<td>-</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>normalize_type</td>
<td>character</td>
<td>tree</td>
<td>tree, forest</td>
<td></td>
</tr>
<tr>
<td>nthread</td>
<td>integer</td>
<td>1</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>ntree_limit</td>
<td>integer</td>
<td>NULL</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>num_parallel_tree</td>
<td>integer</td>
<td>1</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>objective</td>
<td>untyped</td>
<td>&quot;binary:logistic&quot;</td>
<td>TRUE, FALSE</td>
<td></td>
</tr>
<tr>
<td>one_drop</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td></td>
</tr>
<tr>
<td>outputmargin</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td></td>
</tr>
<tr>
<td>predcontrib</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td></td>
</tr>
</tbody>
</table>
Super classes

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

Active bindings

- `internal_valid_scores` (named list() or NULL) The validation scores extracted from model$evaluation_log. If early stopping is activated, this contains the validation scores of the model for the optimal `nrounds`, otherwise the `nrounds` for the final model.
- `internal_tuned_values` (named list() or NULL) If early stopping is activated, this returns a list with `nrounds`, which is extracted from $best_iteration of the model and otherwise NULL.
- `validate` (numeric(1) or character(1) or NULL) How to construct the internal validation data. This parameter can be either NULL, a ratio, "test", or "predefined".

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$clone(deep = FALSE)
Arguments:
dee$ 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

See Also
• Package mlr3extralearners for more learners.
• Dictionary of Learners: mlr3::mlr_learners
• as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
• mlr3pipelines to combine learners with pre- and postprocessing steps.
• Extension packages for additional task types:
  – mlr3proba for probabilistic supervised regression and survival analysis.
  – mlr3cluster for unsupervised clustering.
• mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.

Examples

```r
## Not run:
if (requireNamespace("xgboost", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("classif.xgboost")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

  # Score the predictions
  predictions$score()
}
## End(Not run)

## Not run:
# Train learner with early stopping on spam data set
task = tsk("spam")
# use 30 percent for validation
# Set early stopping parameter
learner = lrn("classif.xgboost",
  nrounds = 100,
  early_stopping_rounds = 10,
  validate = 0.3)

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

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

## End(Not run)
```
GLM with Elastic Net Regularization Regression Learner

Description

Generalized linear models with elastic net regularization. Calls \texttt{glmnet::cv.glmnet()} from package \texttt{glmnet}.

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

Dictionary

This \texttt{mlr3::Learner} can be instantiated via the dictionary \texttt{mlr3::mlr_learners} or with the associated sugar function \texttt{mlr3::lrn()}:

\begin{verbatim}
mlr_learners$get("regr.cv_glmnet")
lrn("regr.cv_glmnet")
\end{verbatim}

Meta Information

- Task type: "regr"
- Predict Types: "response"
- Feature Types: "logical", "integer", "numeric"
- Required Packages: \texttt{mlr3, mlr3learners, glmnet}

Parameters

\begin{center}
\begin{tabular}{lllll}
\hline
Id & Type & Default & Levels & Range \\
\hline
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 & NULL & & - \\
gamma & untyped & - & & - \\
grouped & logical & TRUE & TRUE, FALSE & - \\
intercept & logical & TRUE & TRUE, FALSE & - \\
keep & logical & FALSE & TRUE, FALSE & - \\
lambda & untyped & - & & - \\
\hline
\end{tabular}
\end{center}
mlr_learners_regr.cv_glmnet

lambda.min.ratio numeric - [0, 1]
lower.limits untyped - -
maxit integer 100000 [1, ∞)
mmlam integer 5 [1, ∞)
mxit integer 100 [1, ∞)
xmitnr integer 25 [1, ∞)
nfolds integer 10 [3, ∞)
mlambda integer 100 [1, ∞)
offset untyped NULL -
parallel logical FALSE TRUE, FALSE -
penalty.factor untyped - -
pxmax integer - [0, ∞)
pxmin numeric 1e-09 [0, 1]
prec numeric 1e-10 (∞, ∞)
prec.gamma numeric gamma.1se (∞, ∞)
relax logical FALSE TRUE, FALSE -
s numeric lambda.1se [0, ∞)
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.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

See Also
• Package mlr3extralearners for more learners.
• Dictionary of Learners: mlr3::mlr_learners
• as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
• mlr3pipelines to combine learners with pre- and postprocessing steps.
• Extension packages for additional task types:
  – mlr3proba for probabilistic supervised regression and survival analysis.
  – mlr3cluster for unsupervised clustering.
• mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.


Examples
if (requireNamespace("glmnet", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("regr.cv.glmnet")
  print(learner)
# Define a Task
```r
task = tsk("mtcars")
```

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

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

# print the model
```r
print(learner$model)
```

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

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

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

---

**mlr_learners_regr.glmnet**

*GLM with Elastic Net Regularization Regression Learner*

**Description**

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

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

```r
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>alignment</td>
<td>character</td>
<td>lambda</td>
<td>lambda, fraction</td>
<td>-</td>
</tr>
<tr>
<td>alpha</td>
<td>numeric</td>
<td>1</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>big</td>
<td>numeric</td>
<td>9.9e+35</td>
<td></td>
<td>(-∞, ∞)</td>
</tr>
<tr>
<td>devmax</td>
<td>numeric</td>
<td>0.999</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>dfmax</td>
<td>integer</td>
<td>-</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>eps</td>
<td>numeric</td>
<td>1e-06</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>epsnr</td>
<td>numeric</td>
<td>1e-08</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>exact</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>exclude</td>
<td>integer</td>
<td>-</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>exmx</td>
<td>numeric</td>
<td>250</td>
<td></td>
<td>(-∞, ∞)</td>
</tr>
<tr>
<td>family</td>
<td>character</td>
<td>gaussian</td>
<td>gaussian, poisson</td>
<td>-</td>
</tr>
<tr>
<td>fdev</td>
<td>numeric</td>
<td>1e-05</td>
<td></td>
<td>[0, 1]</td>
</tr>
<tr>
<td>gamma</td>
<td>numeric</td>
<td>1</td>
<td></td>
<td>(-∞, ∞)</td>
</tr>
<tr>
<td>grouped</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>intercept</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>keep</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>lambda</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>lambda.min.ratio</td>
<td>numeric</td>
<td>-</td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>lower.limits</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>maxit</td>
<td>integer</td>
<td>100000</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>mnlam</td>
<td>integer</td>
<td>5</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>mxit</td>
<td>integer</td>
<td>100</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>mxitnr</td>
<td>integer</td>
<td>25</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>newoffset</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>nlambda</td>
<td>integer</td>
<td>100</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>offset</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>parallel</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>penalty.factor</td>
<td>untyped</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>pmax</td>
<td>integer</td>
<td>-</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>pmin</td>
<td>numeric</td>
<td>1e-09</td>
<td></td>
<td>[0, 1]</td>
</tr>
</tbody>
</table>
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


See Also

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- mlr3pipelines to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - mlr3proba for probabilistic supervised regression and survival analysis.
  - mlr3cluster for unsupervised clustering.
- mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.


Examples

```r
if (requireNamespace("glmnet", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("regr.glmnet")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

# Score the predictions
predictions$score()
}

---

**Description**

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

**Initial parameter values**

- store_model:
  - See note.

**Dictionary**

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

```r
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**

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>integer</td>
<td>7</td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>distance</td>
<td>numeric</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>kernel</td>
<td>character</td>
<td>optimal</td>
<td>rectangular, triangular, epanechnikov, biweight, triweight, cos, inv, gaussian, rank, optimal</td>
</tr>
<tr>
<td>scale</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>ykernel</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>store_model</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
</tr>
</tbody>
</table>
Super classes

\texttt{mlr3::Learner} -> \texttt{mlr3::LearnerRegr} -> LearnerRegrKKNN

Methods

Public methods:

- \texttt{LearnerRegrKKNN$new()}
- \texttt{LearnerRegrKKNN$clone()}

Method \texttt{new()}: Creates a new instance of this \texttt{R6} class.

Usage:
LearnerRegrKKNN$new()

Method \texttt{clone()}: The objects of this class are cloneable with this method.

Usage:
LearnerRegrKKNN$clone(deep = FALSE)

Arguments:

depth  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, $\text{model}$ returns a list with the following elements:

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

References


See Also

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- mlr3pipelines to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - mlr3proba for probabilistic supervised regression and survival analysis.
  - mlr3cluster for unsupervised clustering.
- mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.


Examples

```r
if (requireNamespace("kknn", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("regr.kknn")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

  # Score the predictions
  predictions$score()
}
```
Kriging Regression Learner

Description

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

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

Dictionary

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

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

Meta Information

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

Parameters

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>bias.correct</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>checkNames</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>coef.cov</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>coef.trend</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>coef.var</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>control</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>cov.compute</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>covtype</td>
<td>character</td>
<td>matern5_2</td>
<td>gauss, matern5_2, matern3_2, exp, powexp</td>
<td>-</td>
</tr>
<tr>
<td>estim.method</td>
<td>character</td>
<td>MLE</td>
<td>MLE, LOO</td>
<td>-</td>
</tr>
<tr>
<td>gr</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>iso</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>jitter</td>
<td>numeric</td>
<td>0</td>
<td>[0, ∞)</td>
<td></td>
</tr>
<tr>
<td>kernel</td>
<td>untyped</td>
<td>NULL</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
### 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


---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>knots</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>light.return</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>lower</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>multistart</td>
<td>integer</td>
<td>1</td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>noise.var</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>nugget</td>
<td>numeric</td>
<td>-</td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>nugget.estim</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>nugget.stability</td>
<td>numeric</td>
<td>0</td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>optim.method</td>
<td>character</td>
<td>BFGS</td>
<td>BFGS, gen</td>
</tr>
<tr>
<td>parinit</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>penalty</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
</tr>
<tr>
<td>scaling</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>se.compute</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>type</td>
<td>character</td>
<td>SK</td>
<td>SK, UK</td>
</tr>
<tr>
<td>upper</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
</tr>
</tbody>
</table>
See Also

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- mlr3pipelines to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - mlr3proba for probabilistic supervised regression and survival analysis.
  - mlr3cluster for unsupervised clustering.
- mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.


Examples

```r
if (requireNamespace("DiceKriging", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("regr.km")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

  # Score the predictions
  predictions$score()
}
```
**Description**

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

**Dictionary**

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

```r
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**

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

depth Whether to make a deep clone.

See Also

- Package `mlr3extralearners` for more learners.
- Dictionary of Learners: `mlr3::mlr_learners`
- `as.data.table(mlr_learners)` for a table of available Learners in the running session (depending on the loaded packages).
- `mlr3pipelines` to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - `mlr3proba` for probabilistic supervised regression and survival analysis.
mlr_learners_regr.nnet

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


Examples

```r
if (requireNamespace("stats", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("regr.lm")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

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

---

**mlr_learners_regr.nnet**

*Neural Network Regression Learner*

**Description**

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

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

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

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

Meta Information

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

Parameters

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hess</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>MaxNWts</td>
<td>integer</td>
<td>1000</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>Wts</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>abstol</td>
<td>numeric</td>
<td>1e-04</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>censored</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>contrasts</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>decay</td>
<td>numeric</td>
<td>0</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>mask</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>maxit</td>
<td>integer</td>
<td>100</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>na.action</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>rang</td>
<td>numeric</td>
<td>0.7</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>reltol</td>
<td>numeric</td>
<td>1e-08</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>size</td>
<td>integer</td>
<td>3</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>skip</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>subset</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>trace</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>formula</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Initial parameter values

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

Custom mlr3 parameters

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

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

Methods

Public methods:

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

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


See Also

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- mlr3pipelines to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - mlr3proba for probabilistic supervised regression and survival analysis.
  - mlr3cluster for unsupervised clustering.
- mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.

Examples

```r
if (requireNamespace("nnet", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("regr.nnet")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

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

---

**mlr_learners_regr.ranger**  
*Ranger Regression Learner*

**Description**

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

**Dictionary**

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

```r
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>numeric</td>
<td>0.5</td>
<td></td>
<td>$(-\infty, \infty)$</td>
</tr>
<tr>
<td>always.split.variables</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>holdout</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>importance</td>
<td>character</td>
<td>-</td>
<td>none, impurity, impurity_corrected, permutation</td>
<td>-</td>
</tr>
<tr>
<td>keep.inbag</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>max.depth</td>
<td>integer</td>
<td>NULL</td>
<td></td>
<td>$[0, \infty)$</td>
</tr>
<tr>
<td>min.bucket</td>
<td>integer</td>
<td>1</td>
<td></td>
<td>$[1, \infty)$</td>
</tr>
<tr>
<td>min.node.size</td>
<td>integer</td>
<td>5</td>
<td></td>
<td>$[1, \infty)$</td>
</tr>
<tr>
<td>minprop</td>
<td>numeric</td>
<td>0.1</td>
<td></td>
<td>$(-\infty, \infty)$</td>
</tr>
<tr>
<td>mtry</td>
<td>integer</td>
<td>-</td>
<td></td>
<td>$[1, \infty)$</td>
</tr>
<tr>
<td>mtry.ratio</td>
<td>numeric</td>
<td>-</td>
<td></td>
<td>$[0, 1]$</td>
</tr>
<tr>
<td>node.stats</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>num.random.splits</td>
<td>integer</td>
<td>1</td>
<td></td>
<td>$[1, \infty)$</td>
</tr>
<tr>
<td>num.threads</td>
<td>integer</td>
<td>1</td>
<td></td>
<td>$[1, \infty)$</td>
</tr>
<tr>
<td>num.trees</td>
<td>integer</td>
<td>500</td>
<td></td>
<td>$[1, \infty)$</td>
</tr>
<tr>
<td>oob.error</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>quantreg</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>regularization.factor</td>
<td>untyped</td>
<td>1</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>regularization.usedepath</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>replace</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>respect.unordered.factors</td>
<td>character</td>
<td>ignore</td>
<td>ignore, order, partition</td>
<td>-</td>
</tr>
<tr>
<td>sample.fraction</td>
<td>numeric</td>
<td>-</td>
<td></td>
<td>$[0, 1]$</td>
</tr>
<tr>
<td>save.memory</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>scale.permutation.importance</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>se.method</td>
<td>character</td>
<td>infjack</td>
<td>jack, infjack</td>
<td>-</td>
</tr>
<tr>
<td>seed</td>
<td>integer</td>
<td>NULL</td>
<td></td>
<td>$(-\infty, \infty)$</td>
</tr>
<tr>
<td>split.select.weights</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>splitrule</td>
<td>character</td>
<td>variance</td>
<td>variance, extratrees, maxstat</td>
<td>-</td>
</tr>
<tr>
<td>verbose</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>write.forest</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>

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.

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


See Also

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- mlr3pipelines to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - mlr3proba for probabilistic supervised regression and survival analysis.
  - mlr3cluster for unsupervised clustering.
- mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.


Examples

```r
if (requireNamespace("ranger", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("regr.ranger")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

  # Score the predictions
  predictions$score()
}
```
mlr_learners_regr.svm  Support Vector Machine

Description

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

Dictionary

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

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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>cachesize</td>
<td>numeric</td>
<td>40</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>coef0</td>
<td>numeric</td>
<td>0</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>cost</td>
<td>numeric</td>
<td>1</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>cross</td>
<td>integer</td>
<td>0</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>degree</td>
<td>integer</td>
<td>3</td>
<td></td>
<td>[1, ∞)</td>
</tr>
<tr>
<td>epsilon</td>
<td>numeric</td>
<td>0.1</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>fitted</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>gamma</td>
<td>numeric</td>
<td>-</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>kernel</td>
<td>character</td>
<td>radial</td>
<td>linear, polynomial, radial, sigmoid</td>
<td>-</td>
</tr>
<tr>
<td>nu</td>
<td>numeric</td>
<td>0.5</td>
<td></td>
<td>(−∞, ∞)</td>
</tr>
<tr>
<td>scale</td>
<td>untyped</td>
<td>TRUE</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>shrinking</td>
<td>logical</td>
<td>TRUE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>tolerance</td>
<td>numeric</td>
<td>0.001</td>
<td></td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>type</td>
<td>character</td>
<td>eps-regression</td>
<td>eps-regression, nu-regression</td>
<td>-</td>
</tr>
</tbody>
</table>

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


See Also


• Package mlr3extralearners for more learners.

• Dictionary of Learners: mlr3::mlr_learners

• as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).

• mlr3pipelines to combine learners with pre- and postprocessing steps.

• Extension packages for additional task types:
  – mlr3proba for probabilistic supervised regression and survival analysis.
  – mlr3cluster for unsupervised clustering.

• mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.

Examples

```r
if (requireNamespace("e1071", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("regr.svm")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

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

mlr_learners_regr.xgboost

**Extreme Gradient Boosting Regression Learner**

**Description**

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


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

**Dictionary**

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

```r
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

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Levels</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha</td>
<td>numeric</td>
<td>0</td>
<td>TRUE, FALSE</td>
<td>[0, ∞)</td>
</tr>
<tr>
<td>approxcontrib</td>
<td>logical</td>
<td>FALSE</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>base_score</td>
<td>numeric</td>
<td>0.5</td>
<td>(-∞, ∞)</td>
<td></td>
</tr>
<tr>
<td>booster</td>
<td>character</td>
<td>gbtree</td>
<td>gbtree, gblinear, dart</td>
<td>-</td>
</tr>
<tr>
<td>callbacks</td>
<td>untyped</td>
<td>list()</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>colsample_bylevel</td>
<td>numeric</td>
<td>1</td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>colsample_bynode</td>
<td>numeric</td>
<td>1</td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>colsample_bytree</td>
<td>numeric</td>
<td>1</td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>device</td>
<td>untyped</td>
<td>&quot;cpu&quot;</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>disable_default_eval_metric</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>early_stopping_rounds</td>
<td>integer</td>
<td>NULL</td>
<td>[1, ∞)</td>
<td></td>
</tr>
<tr>
<td>eta</td>
<td>numeric</td>
<td>0.3</td>
<td>[0, 1]</td>
<td></td>
</tr>
<tr>
<td>eval_metric</td>
<td>untyped</td>
<td>&quot;rmse&quot;</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>feature_selector</td>
<td>character</td>
<td>cyclic</td>
<td>cyclic, shuffle, random, greedy, thrifty</td>
<td>-</td>
</tr>
<tr>
<td>feval</td>
<td>untyped</td>
<td>NULL</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>gamma</td>
<td>numeric</td>
<td>0</td>
<td>[0, ∞)</td>
<td></td>
</tr>
<tr>
<td>grow_policy</td>
<td>character</td>
<td>depthwise</td>
<td>depthwise, lossguide</td>
<td>-</td>
</tr>
<tr>
<td>interaction_constraints</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>iterationrange</td>
<td>untyped</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>lambda</td>
<td>numeric</td>
<td>1</td>
<td>[0, ∞)</td>
<td></td>
</tr>
<tr>
<td>lambda_bias</td>
<td>numeric</td>
<td>0</td>
<td>[0, ∞)</td>
<td></td>
</tr>
<tr>
<td>max_bin</td>
<td>integer</td>
<td>256</td>
<td>[2, ∞)</td>
<td></td>
</tr>
<tr>
<td>max_delta_step</td>
<td>numeric</td>
<td>0</td>
<td>[0, ∞)</td>
<td></td>
</tr>
<tr>
<td>max_depth</td>
<td>integer</td>
<td>6</td>
<td>[0, ∞)</td>
<td></td>
</tr>
<tr>
<td>max_leaves</td>
<td>integer</td>
<td>0</td>
<td>[0, ∞)</td>
<td></td>
</tr>
<tr>
<td>maximize</td>
<td>logical</td>
<td>NULL</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>min_child_weight</td>
<td>numeric</td>
<td>1</td>
<td>[0, ∞)</td>
<td></td>
</tr>
<tr>
<td>missing</td>
<td>numeric</td>
<td>NA</td>
<td>(-∞, ∞)</td>
<td></td>
</tr>
<tr>
<td>monotone_constraints</td>
<td>untyped</td>
<td>0</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>normalize_type</td>
<td>character</td>
<td>tree</td>
<td>tree, forest</td>
<td>-</td>
</tr>
<tr>
<td>nrounds</td>
<td>integer</td>
<td>-</td>
<td>[1, ∞)</td>
<td></td>
</tr>
<tr>
<td>nthread</td>
<td>integer</td>
<td>1</td>
<td>[1, ∞)</td>
<td></td>
</tr>
<tr>
<td>ntreelimit</td>
<td>integer</td>
<td>NULL</td>
<td>[1, ∞)</td>
<td></td>
</tr>
<tr>
<td>num_parallel_tree</td>
<td>integer</td>
<td>1</td>
<td>[1, ∞)</td>
<td></td>
</tr>
<tr>
<td>objective</td>
<td>untyped</td>
<td>&quot;reg:squarederror&quot;</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>one_drop</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>outputmargin</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
<tr>
<td>predcontrib</td>
<td>logical</td>
<td>FALSE</td>
<td>TRUE, FALSE</td>
<td>-</td>
</tr>
</tbody>
</table>
Early Stopping and Validation

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

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
```

Active bindings

- **internal_valid_scores**: (named list() or NULL) The validation scores extracted from `model$evaluation_log`. If early stopping is activated, this contains the validation scores of the model for the optimal `nrounds`, otherwise the `nrounds` for the final model.
- **internal_tuned_values**: (named list() or NULL) If early stopping is activated, this returns a list with `nrounds`, which is extracted from `$best_iteration` of the model and otherwise NULL.
- **validate**: (numeric(1) or character(1) or NULL) How to construct the internal validation data. This parameter can be either NULL, a ratio, "test", or "predefined". Returns the `$best_iteration` when early stopping is activated.

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


See Also

- Package mlr3extralearners for more learners.
- Dictionary of Learners: mlr3::mlr_learners
- as.data.table(mlr_learners) for a table of available Learners in the running session (depending on the loaded packages).
- mlr3pipelines to combine learners with pre- and postprocessing steps.
- Extension packages for additional task types:
  - mlr3proba for probabilistic supervised regression and survival analysis.
  - mlr3cluster for unsupervised clustering.
- mlr3tuning for tuning of hyperparameters, mlr3tuningspaces for established default tuning spaces.


Examples

```r
## Not run:
if (requireNamespace("xgboost", quietly = TRUE)) {
  # Define the Learner and set parameter values
  learner = lrn("regr.xgboost")
  print(learner)

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

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

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

  # print the model
  print(learner$model)

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

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

# Score the predictions
predictions$score()
}

## End(Not run)

## Not run:
# Train learner with early stopping on spam data set

# use 30 percent for validation
# Set early stopping parameter
learner = lrn("regr.xgboost",
    nrounds = 100,
    early_stopping_rounds = 10,
    validate = 0.3
)

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

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

## End(Not run)
Index

* Learner
  - mlr_learners_classif.cv_glmnet, 3
  - mlr_learners_classif.glmnet, 6
  - mlr_learners_classif.kknn, 10
  - mlr_learners_classif.lda, 13
  - mlr_learners_classif.log_reg, 15
  - mlr_learners_classif.multinom, 18
  - mlr_learners_classif.naive_bayes, 21
  - mlr_learners_classif.nnet, 23
  - mlr_learners_classif.qda, 26
  - mlr_learners_classif.ranger, 29
  - mlr_learners_classif.svm, 32
  - mlr_learners_classif.xgboost, 35
  - mlr_learners_regr.cv_glmnet, 40
  - mlr_learners_regr.glmnet, 43
  - mlr_learners_regr.kknn, 47
  - mlr_learners_regr.lm, 53
  - mlr_learners_regr.nnet, 55
  - mlr_learners_regr.ranger, 58
  - mlr_learners_regr.svm, 62
  - mlr_learners_regr.xgboost, 64

  contr.poly(), 17, 54
  contr.treatment(), 17, 54

  DiceKriging::km(), 50
  Dictionary, 5, 9, 12, 14, 17, 20, 22, 25, 28, 31, 34, 38, 42, 46, 49, 52, 54, 57, 61, 63, 68
  dictionary, 3, 7, 10, 13, 16, 19, 21, 24, 26, 29, 32, 35, 40, 44, 47, 50, 53, 56, 58, 62, 64

  e1071::naiveBayes(), 21
  e1071::svm(), 32, 62

  glmnet::cv.glmnet(), 3, 7, 40, 43
  glmnet::glmnet(), 7, 43
  glmnet::predict.glmnet(), 5, 9, 41, 45
  knn::knn(), 10–12, 47, 48

  LearnerClassifCVGlmnet
    (mlr_learners_classif.cv_glmnet), 3
  LearnerClassifGlmnet
    (mlr_learners_classif.glmnet), 6
  LearnerClassifKKNN
    (mlr_learners_classif.kknn), 10
  LearnerClassifLDA
    (mlr_learners_classif.lda), 13
  LearnerClassifLogReg
    (mlr_learners_classif.log_reg), 15
  LearnerClassifMultinom
    (mlr_learners_classif.multinom), 18
  LearnerClassifNaiveBayes
    (mlr_learners_classif.naive_bayes), 21
  LearnerClassifNnet
    (mlr_learners_classif.nnet), 23
  LearnerClassifQDA
    (mlr_learners_classif.qda), 26
  LearnerClassifRanger
    (mlr_learners_classif.ranger), 29
  LearnerClassifSVM
    (mlr_learners_classif.svm), 32
  LearnerClassifXgboost
    (mlr_learners_classif.xgboost), 35
  LearnerRegrCVGlmnet
    (mlr_learners_regr.cv_glmnet), 40
  LearnerRegrGlmnet
    (mlr_learners_regr.glmnet), 43
LearnerRegrKNN
(\texttt{mlr_learners_regr.kknn}), 47
LearnerRegrKM (\texttt{mlr_learners_regr.km}), 50
LearnerRegrLM (\texttt{mlr_learners_regr.lm}), 53
LearnerRegrNnet
(\texttt{mlr_learners_regr.nnet}), 55
LearnerRegrRanger
(\texttt{mlr_learners_regr.ranger}), 58
LearnerRegrSVM (\texttt{mlr_learners_regr.svm}), 62
LearnerRegrXgboost
(\texttt{mlr_learners_regr.xgboost}), 64
Learnrers, 5, 6, 9, 12, 14, 17, 20, 22, 25, 28, 31, 34, 38, 42, 46, 49, 52, 54, 57, 61, 63, 68

\texttt{MASS::lda()}, 13
\texttt{MASS::lda()}, 26
\texttt{mlr3::Learner}, 3, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, 21, 22, 24–27, 29, 30, 32, 33, 35, 37, 40, 41, 44, 45, 47, 48, 50, 51, 53, 54, 56–58, 60, 62, 64, 66, 67
\texttt{mlr3::Learner}, 5, 8, 11, 14, 17, 19, 22, 25, 27, 30, 33, 37
\texttt{mlr3::LearnerRegr}, 41, 45, 48, 51, 54, 57, 60, 62, 67
\texttt{mlr3::lrm()}, 3, 7, 10, 13, 16, 19, 21, 24, 26, 29, 32, 35, 40, 44, 47, 50, 53, 56, 58, 62, 64
\texttt{mlr3::mlr_learners}, 3, 5, 7, 9, 10, 12–14, 16, 17, 19–22, 24–26, 28, 29, 31, 32, 34, 35, 38, 40, 42, 44, 46, 47, 49, 50, 52–54, 56–58, 61–64, 68
\texttt{mlr3learnre (mlr3learners-package)}, 2
\texttt{mlr3learners-package}, 2
\texttt{mlr_learners_classif.cv_glmnet}, 3, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_classif.cv_glmnet}, 7, 43
\texttt{mlr_learners_classif.glmnet}, 6, 6, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_classif.kknn}, 6, 9, 10, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_classif.lda}, 6, 9, 12, 13, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_classif.log_reg}, 6, 9, 12, 15, 16, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_classif.multinom}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_classif.naive_bayes}, 6, 9, 12, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_classif.nnet}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_classif.ranger}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_classif.svm}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 32, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_classif.xgboost}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 40, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_regr.cv_glmnet}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_regr.cv.glmnet()}, 7, 43
\texttt{mlr_learners_regr.glmnet}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 43, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_regr.kknn}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 47, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_regr.km}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_regr.lm}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_regr.nnet}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_regr.ranger}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
\texttt{mlr_learners_regr.svm}, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 68
mlr_learners_regr.xgboost, 6, 9, 12, 15, 18, 20, 23, 25, 28, 31, 34, 38, 42, 46, 49, 52, 55, 57, 61, 63, 64

nnet::multinom(), 18
nnet::nnet.formula(), 23, 55

R6, 5, 8, 11, 14, 17, 20, 22, 23, 27, 30, 33, 38, 41, 45, 48, 51, 54, 57, 60, 63, 67

ranger::ranger(), 29, 58

stats::glm(), 5, 8, 15, 16
stats::lm(), 53
stats::logLik(), 17, 20, 54

xgboost::xgb.importance(), 38, 67
xgboost::xgb.train(), 35, 64