Package ‘mlr3learners’

August 29, 2020

Title  Recommended Learners for ‘mlr3’
Version 0.3.0
Description  Recommended Learners for ‘mlr3’. Extends ‘mlr3’ and ‘mlr3proba’ with interfaces to essential machine learning packages on CRAN. This includes, but is not limited to: (penalized) linear and logistic regression, linear and quadratic discriminant analysis, k-nearest neighbors, naive Bayes, support vector machines, and gradient boosting.
License LGPL-3
BugReports https://github.com/mlr-org/mlr3learners/issues
Depends R (>= 3.1.0)
Imports data.table, mlr3 (>= 0.5.0), mlr3misc (>= 0.5.0), paradox, R6
Suggests bitext, checkmate, DiceKriging, distr6, e1071, glmnet, knn, knitr, lgr, MASS, mlr3proba (>= 0.2.1), nnet, pracma, ranger, rgenoud, rmarkdown, testthat, xgboost
RdMacros mlr3misc
Encoding UTF-8
NeedsCompilation no
RoxygenNote 7.1.1
Author Michel Lang [cre, aut] (<https://orcid.org/0000-0001-9754-0393>), Quay Au [aut] (<https://orcid.org/0000-0002-5252-8902>), Stefan Coors [aut] (<https://orcid.org/0000-0002-7465-2146>), Patrick Schratz [aut] (<https://orcid.org/0000-0003-0748-6624>)
Maintainer Michel Lang <michellang@gmail.com>
Repository CRAN
Date/Publication 2020-08-29 19:10:02 UTC
R topics documented:

mlr3learners-package .................................................. 2
mlr_learners_classif.cv_glmnet ........................................ 3
mlr_learners_classif.glmnet ........................................... 4
mlr_learners_classif.kknn ............................................. 5
mlr_learners_classif.lda .............................................. 7
mlr_learners_classif.log_reg ......................................... 8
mlr_learners_classif.multinom ........................................ 10
mlr_learners_classif.naive_bayes .................................... 11
mlr_learners_classif.qda ............................................. 12
mlr_learners_classif.ranger ......................................... 13
mlr_learners_classif.svm ............................................. 14
mlr_learners_classif.xgboost ....................................... 16
mlr_learners_regr.cv_glmnet ......................................... 17
mlr_learners_regr.glmnet ............................................. 19
mlr_learners_regr.kknn .............................................. 20
mlr_learners_regr.km ................................................ 21
mlr_learners_regr.lm ................................................ 23
mlr_learners_regr.ranger ............................................ 24
mlr_learners_regr.svm ............................................. 25
mlr_learners_regr.xgboost ......................................... 27
mlr_learners_surv.cv_glmnet ......................................... 28
mlr_learners_surv.glmnet ........................................... 30
mlr_learners_surv.ranger ........................................... 31
mlr_learners_surv.xgboost .......................................... 32

Index 35

mlr3learners-package  mlr3learners: Recommended Learners for 'mlr3'

Description


Author(s)

Maintainer: Michel Lang <michellang@gmail.com> (ORCID)
Authors:

• Quay Au <quayau@gmail.com> (ORCID)
• Stefan Coors <mail@stefancoors.de> (ORCID)
• Patrick Schratz <patrick.schratz@gmail.com> (ORCID)
**Description**

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

**Dictionary**

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

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

**Super classes**

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

**Methods**

**Public methods:**

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

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

*Usage:*

```r
LearnerClassifCVGlmnet$new()
```

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

*Usage:*

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

*Arguments:*

deep Whether to make a deep clone.
References


See Also

*Dictionary of Learners: mlr3::mlr_learners*

Examples

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

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

---

**mlr_learners_classif.glmnet**

*GLM with Elastic Net Regularization Classification Learner*

Description

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

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

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

Dictionary

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

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

Super classes

`mlr3::Learner < mlr3::LearnerClassif < LearnerClassifGlmnet`
Methods

Public methods:

• LearnerClassifGlmnet$new()
• LearnerClassifGlmnet$clone()

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

Usage:
LearnerClassifGlmnet$new()

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

Usage:
LearnerClassifGlmnet$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

References


See Also

Dictionary of Learners: mlr3::mlr_learners

Examples

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

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

mlr_learners_classif.kknn

k-Nearest-Neighbor Classification Learner

Description

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

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

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

Super classes

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

Methods

**Public methods:**

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

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

*Usage:*

```r
LearnerClassifKKNN$new()
```

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

*Usage:*

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

*Arguments:*

- `deep` Whether to make a deep clone.

Note

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

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

References


**mlr_learners_classif.lda**

**See Also**

Dictionary of Learners: mlr3::mlr_learners

**Examples**

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

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

---

**Description**

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

**Details**

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

**Dictionary**

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

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

**Super classes**

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

**Methods**

**Public methods:**

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

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

*Usage:*
LearnerClassifLDA$new()

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

Usage:
LearnerClassifLDA$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

References

See Also
Dictionary of Learners: mlr3::mlr_learners

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

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

mlr_learners_classif.log_reg

Logistic Regression Classification Learner

Description
Classification via logistic regression. Calls stats::glm() with family set to "binomial". Argument model is set to FALSE.

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

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

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

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

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

Super classes

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

Methods

Public methods:

• `LearnerClassifLogReg$new()`
• `LearnerClassifLogReg$clone()`

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

Usage:

`LearnerClassifLogReg$new()`

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

Usage:

`LearnerClassifLogReg$clone(deep = FALSE)`

Arguments:

depth  Whether to make a deep clone.

See Also

Dictionary of Learners: `mlr3::mlr_learners`

Examples

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

  # available parameters:
  learner$param_set$ids()
}
```
**mlr_learners_classif.multinom**

*Multinomial log-linear learner via neural networks*

---

**Description**


**Dictionary**

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

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

**Super classes**

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

**Methods**

**Public methods:**

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

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

*Usage:*

```
LearnerClassifMultinom$new()
```

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

*Usage:*

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

*Arguments:*

- `deep` Whether to make a deep clone.

**See Also**

Dictionary of Learners: `mlr3::mlr_learners`

**Examples**

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

  # available parameters:
  learner$param_set$ids()
}
```
Naive Bayes Classification Learner

Description

Naive Bayes classification. Calls \texttt{e1071::naiveBayes()} from package \texttt{e1071}.

Dictionary

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

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

Super classes

\texttt{mlr3::Learner} -> \texttt{mlr3::LearnerClassif} -> LearnerClassifNaiveBayes

Methods

Public methods:

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

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

Usage:

\begin{verbatim}
LearnerClassifNaiveBayes$new()
\end{verbatim}

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

Usage:

\begin{verbatim}
LearnerClassifNaiveBayes$clone(deep = FALSE)
\end{verbatim}

Arguments:

deep \hspace{1em} Whether to make a deep clone.

See Also

Dictionary of Learners: \texttt{mlr3::mlr_learners}

Examples

\begin{verbatim}
if (requireNamespace("e1071")) {
  learner = mlr3::lrn("classif.naive_bayes")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
\end{verbatim}
Quadratic Discriminant Analysis Classification Learner

Description

Quadratic discriminant analysis. Calls \texttt{MASS::qda()} from package \texttt{MASS}.

Details

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

Dictionary

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

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

Super classes

\texttt{mlr3::Learner} \rightarrow \texttt{mlr3::LearnerClassif} \rightarrow \texttt{LearnerClassifQDA}

Methods

Public methods:
- \texttt{LearnerClassifQDA$new()}
- \texttt{LearnerClassifQDA$clone()}

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

Usage:

\begin{verbatim}
LearnerClassifQDA$new()
\end{verbatim}

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

Usage:

\begin{verbatim}
LearnerClassifQDA$clone(deep = FALSE)
\end{verbatim}

Arguments:
- deep Whether to make a deep clone.

References

Ranger Classification Learner

Description

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

Dictionary

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

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

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

Super classes

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

Methods

Public methods:

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

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

Usage:

```r
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:
depth Whether to make a deep clone.

References


See Also
Dictionary of Learners: mlr3::mlr_learners

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

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

m Wilkinson

Support Vector Machine

Description
A learner for a classification support vector machine implemented in e1071::svm().
Dictionary

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

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

Super classes

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

Methods

Public methods:

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

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

Usage:

```r
LearnerClassifSVM$new()
```

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

Usage:

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

Arguments:

depth Whether to make a deep clone.

References


See Also

Dictionary of Learners: `mlr3::mlr_learners`

Examples

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

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

Extreme Gradient Boosting Classification Learner

Description

eXtreme Gradient Boosting classification. Calls \texttt{xgboost::xgb.train()} from package \texttt{xgboost}.

Custom mlr3 defaults

- \texttt{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.

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

Dictionary

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

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

Super classes

\texttt{mlr3::Learner} -> \texttt{mlr3::LearnerClassif} -> LearnerClassifXgboost

Methods

Public methods:

- \texttt{LearnerClassifXgboost$new()}
- \texttt{LearnerClassifXgboost$importance()}
- \texttt{LearnerClassifXgboost$clone()}

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

\textit{Usage}:

\texttt{LearnerClassifXgboost$new()}

Method \texttt{importance()}: The importance scores are calculated with \texttt{xgboost::xgb.importance()}.

\textit{Usage}:
LearnerClassifXgboost$importance()

Returns: Named numeric().

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

Usage:
LearnerClassifXgboost$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

References

See Also
Dictionary of Learners: mlr3::mlr_learners

Examples

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

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

mlr_learners_regr.cv_glmnet

GLM with Elastic Net Regularization Regression Learner

Description
Generalized linear models with elastic net regularization. Calls glmnet::cv.glmnet() from package glmnet.
The default for hyperparameter family is changed to "gaussian".

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

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

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

Methods

**Public methods:**

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

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

*Usage:*

```
LearnerRegrCVGlmnet$new()
```

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

*Usage:*

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

*Arguments:*

- `deep`: Whether to make a deep clone.

References


See Also

Dictionary of Learners: `mlr3::mlr_learners`

Examples

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

  # available parameters:
  learner$param_set$ids()
}
```
**mlr_learners_regr.glmnet**

---

**GLM with Elastic Net Regularization Regression Learner**

---

**Description**

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

The default for hyperparameter `family` is changed to "gaussian".

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

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

**Dictionary**

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

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

**Super classes**

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

**Methods**

**Public methods:**

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

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

*Usage:*

```
LearnerRegrGlmnet$new()
```

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

*Usage:*

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

*Arguments:*

- `deep` Whether to make a deep clone.

**References**

mnr_learners_regr.kknn

k-Nearest-Neighbor Regression Learner

Description

k-Nearest-Neighbor regression. Calls \texttt{kknn::kknn()} from package \texttt{kknn}.

Dictionary

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

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

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.

\textit{Usage}:

\texttt{LearnerRegrKKNN\$new()}

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

\textit{Usage}:

\texttt{LearnerRegrKKNN\$clone(deep = FALSE)}

\textit{Arguments}:

deepl Whether to make a deep clone.
Note

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

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

References


See Also

Dictionary of Learners: mlr3::mlr_learners

Examples

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

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

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

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

Super classes

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

Methods

Public methods:

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

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

**Usage**:

```r
LearnerRegrKM$new()
```

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

**Usage**:

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

**Arguments**:

- `deep` Whether to make a deep clone.

References


See Also

Dictionary of Learners: mlr3::mlr_learners

Examples

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

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

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

**Dictionary**

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

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

```r
lrn("regr.lm")
```

**Contrasts**

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

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

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

**Super classes**

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

**Methods**

**Public methods:**

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

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

```
Usage:
LearnerRegrLM$new()
```

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

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

**Arguments:**

depth Whether to make a deep clone.
mlr_learners_regr.ranger

Ranger Regression Learner

Description

Random regression forest. Calls \texttt{ranger::ranger()} from package \texttt{ranger}.

Dictionary

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

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

Super classes

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

Methods

Public methods:

- \texttt{LearnerRegrRanger$new()}
- \texttt{LearnerRegrRanger$importance()}
- \texttt{LearnerRegrRanger$ooe_error()}
- \texttt{LearnerRegrRanger$clone()}

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

Usage:

\begin{verbatim}
LearnerRegrRanger$new()
\end{verbatim}

Method \texttt{importance()}: The importance scores are extracted from the model slot \texttt{variable.importance}. Parameter \texttt{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:
deepl Whether to make a deep clone.

References


See Also

Dictionary of Learners: mlr3::mlr_learners

Examples

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

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

A learner for a regression support vector machine implemented in e1071::svm().
Dictionary

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

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

Super classes

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

Methods

Public methods:

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

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

Usage:

```r
LearnerRegrSVM$new()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

References


See Also

Dictionary of Learners: `mlr3::mlr_learners`

Examples

```r
if (requireNamespace("e1071")) {
  learner = mlr3::lrn("regr.svm")
  print(learner)

  # available parameters:
  learner$param_set$ids()
}
```
**Extreme Gradient Boosting Regression Learner**

**Description**

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

**Custom mlr3 defaults**

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

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

**Dictionary**

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

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

**Super classes**

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

**Methods**

**Public methods:**

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

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

**Usage:**

```
LearnerRegrXgboost$new()
```

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

**Usage:**
LearnerRegrXgboost$importance()

*Returns:* Named numeric().

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

*Usage:*

LearnerRegrXgboost$clone(deep = FALSE)

*Arguments:*

deep Whether to make a deep clone.

**References**


**See Also**

Dictionary of Learners: mlr3::mlr_learners

**Examples**

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

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

**Dictionary**

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

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

**Description**

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

The default for hyperparameter `family` is changed to "cox".

**Cross-Validated GLM with Elastic Net Regularization Survival Learner**
Super classes

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

Methods

**Public methods:**

- `LearnerSurvCVGlmnet$new()`
- `LearnerSurvCVGlmnet$clone()`

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

*Usage:*

```
LearnerSurvCVGlmnet$new()
```

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

*Usage:*

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

*Arguments:*

depth Whether to make a deep clone.

References


See Also

Dictionary of Learners: `mlr3::mlr_learners`

Examples

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

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

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

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

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

Dictionary

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

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

Super classes

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

Methods

**Public methods:**

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

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

*Usage:*

```r
LearnerSurvGlmnet$new()
```

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

*Usage:*

```r
LearnerSurvGlmnet$clone(deep = FALSE)
```

*Arguments:*

deep Whether to make a deep clone.

References

mlr_learners_surv.ranger

See Also

Dictionary of Learners: mlr3::mlr_learners

Examples

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

  # available parameters:
  learner$param_set$id$s()}
```

---

**mlr_learners_surv.ranger**

*Ranger Survival Learner*

**Description**

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

**Dictionary**

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

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

**Super classes**

`mlr3::Learner` $\rightarrow$ `mlr3proba::LearnerSurv` $\rightarrow$ `LearnerSurvRanger`

**Methods**

**Public methods:**

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

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

**Usage:**

`LearnerSurvRanger$new()`

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

Returns: Named numeric().

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

Returns: numeric(1).

Method clone(): The objects of this class are cloneable with this method.
Usage:
LearnerSurvRanger$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

References

See Also
Dictionary of Learners: mlr::mlr_learners

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

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

mlr_learners_surv.xgboost

Extreme Gradient Boosting Survival Learner

Description
eXtreme Gradient Boosting regression. Calls xgboost::xgb.train() from package xgboost.
**Custom mlr3 defaults**

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

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

- **objective**:
  - Actual default: reg:squarederror
  - Adjusted default: survival:cox
  - Reason for change: This is the only available objective for survival.

- **eval_metric**:
  - Actual default: no default
  - Adjusted default: cox-nloglik
  - Reason for change: Only sensible metric for objective.

**Dictionary**

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

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

**Super classes**

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

**Methods**

- **Public methods:**
  - LearnerSurvXgboost$new()
  - LearnerSurvXgboost$importance()
  - LearnerSurvXgboost$clone()

- **Method new()**: Creates a new instance of this R6 class.
  
  **Usage:**
  LearnerSurvXgboost$new()

- **Method importance()**: The importance scores are calculated with xgboost::xgb.importance().
  
  **Usage:**
LearnerSurvXgboost$importance()

*Returns*: Named numeric().

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

**Usage**:

LearnerSurvXgboost$clone(deep = FALSE)

**Arguments**:

depth Whether to make a deep clone.

**References**


**See Also**

Dictionary of Learners: mlr3::mlr_learners

**Examples**

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

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

contr.poly(), 9, 23
contr.treatment(), 9, 23

DiceKriging::km(), 21
dictionary, 3, 4, 6–8, 10–13, 15–17, 19, 20, 22–24, 26–28, 30, 31, 33

e1071::naiveBayes(), 11
e1071::svm(), 14, 25

glmnet::cv.glmnet(), 3, 17, 28
glmnet::glmnet(), 4, 19, 30

kknn::knn(), 5, 6, 20, 21

Learner, 3, 4, 6–8, 10–13, 15–17, 19, 20, 22–24, 26–28, 30, 31, 33
LearnerClassifCVGlmnet
(mlr_learners_classif.cv_glmnet), 3
LearnerClassifGlmnet
(mlr_learners_classif.glmnet), 4
LearnerClassifKNN
(mlr_learners_classif.knn), 5
LearnerClassifLDA
(mlr_learners_classif.lda), 7
LearnerClassifLogReg
(mlr_learners_classif.log_reg), 8
LearnerClassifMultinom
(mlr_learners_classif.multinom), 10
LearnerClassifNaiveBayes
(mlr_learners_classif.naive_bayes), 11
LearnerClassifQDA
(mlr_learners_classif.qda), 12

LearnerClassifRanger
(mlr_learners_classif.ranger), 13
LearnerClassifSVM
(mlr_learners_classif.svm), 14
LearnerClassifXgboost
(mlr_learners_classif.xgboost), 16
LearnerRegrCVGlmnet
(mlr_learners_regr.cv_glmnet), 17
LearnerRegrGlmnet
(mlr_learners_regr.glmnet), 19
LearnerRegrKNN
(mlr_learners_regr.knn), 20
LearnerRegrKM (mlr_learners_regr.km), 21
LearnerRegrLM (mlr_learners_regr.lm), 23
LearnerRegrRanger
(mlr_learners_regr.ranger), 24
LearnerRegrSVM (mlr_learners_regr.svm), 25
LearnerRegrXgboost
(mlr_learners_regr.xgboost), 27
Learners, 4, 5, 7–11, 13–15, 17, 18, 20–22, 24–26, 28, 29, 31, 32, 34
LearnerSurvCVGlmnet
(mlr_learners_surv.cv_glmnet), 28
LearnerSurvGlmnet
(mlr_learners_surv.glmnet), 30
LearnerSurvRanger
(mlr_learners_surv.ranger), 31
LearnerSurvXgboost
(mlr_learners_surv.xgboost), 32

lrn(), 3, 4, 6–8, 10–13, 15–17, 19, 20, 22–24, 26–28, 30, 31, 33

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

nnet::multinom(), 10

R6, 3, 5–7, 9–13, 15, 16, 18–20, 22–24, 26, 27, 29–31, 33
ranger::ranger(), 13, 24, 31
stats::glm(), 8
stats::lm(), 23
xgboost::xgb.importance(), 16, 27, 33
xgboost::xgb.train(), 16, 27, 32