Package ‘mlr3’

March 5, 2021

Title Machine Learning in R - Next Generation

Version 0.11.0

Description Efficient, object-oriented programming on the building blocks of machine learning. Provides 'R6' objects for tasks, learners, resamplings, and measures. The package is geared towards scalability and larger datasets by supporting parallelization and out-of-memory data-backends like databases. While 'mlr3' focuses on the core computational operations, add-on packages provide additional functionality.

License LGPL-3


Depends R (>= 3.1.0)

Imports R6 (>= 2.4.1), backports, checkmate (>= 2.0.0), data.table (>= 1.13.6), digest, future.apply (>= 1.5.0), lgr (>= 0.3.4), mlbench, mlr3measures (>= 0.3.0), mlr3misc (>= 0.7.0), parallelly, palmerpenguins, paradox (>= 0.6.0), uuid

Suggests Matrix, callr, datasets, distr6, evaluate, future, future.callr, mlr3data, progressr, rpart, testthat (>= 3.0.0)

Encoding UTF-8

LazyData true

Config/testthat/edition 3

Config/testthat/parallel false

NeedsCompilation no

RoxygenNote 7.1.1

Collate 'mlr_reflections.R' 'BenchmarkResult.R' 'DataBackend.R'
 'DataBackendCbind.R' 'DataBackendDataTable.R'
 'DataBackendMatrix.R' 'DataBackendRbind.R'
 'DataBackendRename.R' 'Learner.R' 'LearnerClassif.R'
 'mlr_learners.R' 'LearnerClassifDebug.R'
 'LearnerClassifFeatureless.R' 'LearnerClassifRpart.R'
Author  Michel Lang [cre, aut] (<https://orcid.org/0000-0001-9754-0393>),
Bernd Bischl [aut] (<https://orcid.org/0000-0001-6002-6980>),
Jakob Richter [aut] (<https://orcid.org/0000-0003-4481-5554>),
Patrick Schratz [aut] (<https://orcid.org/0000-0003-0748-6624>),
Giuseppe Casalicchio [ctb] (<https://orcid.org/0000-0001-5324-5966>),
Stefan Coors [ctb] (<https://orcid.org/0000-0002-7465-2146>),
Quay Au [ctb] (<https://orcid.org/0000-0002-5252-8902>),
Martin Binder [aut],
Marc Becker [ctb] (<https://orcid.org/0000-0002-8115-0400>)

Maintainer  Michel Lang <michellang@gmail.com>

Repository  CRAN

Date/Publication  2021-03-05 14:00:06 UTC
**R topics documented:**

- `mlr3-package` .................................................. 6
- `as_benchmark_result` ......................................... 8
- `as_data_backend.Matrix` ....................................... 8
- `as_learner` ...................................................... 9
- `as_measure` ..................................................... 10
- `as_prediction` .................................................. 11
- `as_prediction_classif` ......................................... 12
- `as_prediction_data` ............................................. 13
- `as_prediction_regr` ............................................ 14
- `as_resample_result` ............................................ 15
- `as_resampling` .................................................. 16
- `as_result_data` ................................................ 16
- `as_task` .......................................................... 18
- `as_task_classif` ............................................... 18
- `as_task_regr` ................................................... 20
- `benchmark` ....................................................... 20
- `BenchmarkResult` ............................................... 24
- `benchmark_grid` ................................................ 29
- `convert_task` ................................................... 30
- `DataBackend` ..................................................... 31
- `DataBackendDataTable` .......................................... 32
- `DataBackendMatrix` ............................................. 35
- `default_measures` ............................................... 37
- `Learner` .......................................................... 38
- `LearnerClassif` ................................................. 44
- `LearnerRegr` .................................................... 46
- `Measure` .......................................................... 48
- `MeasureClassif` ................................................ 52
- `MeasureRegr` .................................................... 54
- `mlr_learners` ................................................... 56
- `mlr_learners_classif.debug` .................................. 57
- `mlr_learners_classif.featureless` ............................ 59
- `mlr_learners_classif.rpart` ................................... 61
- `mlr_learners_regr.featureless` ................................. 63
- `mlr_learners_regr.rpart` ...................................... 64
- `mlr_measures` .................................................... 66
- `mlr_measures_classif.acc` ..................................... 67
- `mlr_measures_classif.auc` ..................................... 68
- `mlr_measures_classif.bacc` .................................... 69
- `mlr_measures_classif.bbrier` .................................. 70
- `mlr_measures_classif.ce` ....................................... 72
- `mlr_measures_classif.costs` ................................... 73
- `mlr_measures_classif.dor` ...................................... 75
- `mlr_measures_classif.fbeta` ................................... 76
- `mlr_measures_classif.fdr` ...................................... 77
- `mlr_measures_classif.fn` ....................................... 78
### R topics documented:

```
mkr_measures_classif.fnr                      . 80
mlr_measures_classif.fomr                     . 81
mlr_measures_classif.fp                      . 82
mlr_measures_classif.fpr                     . 83
mlr_measures_classif.logloss                 . 84
mlr_measures_classif.mbrier                  . 85
mlr_measures_classif.mcc                     . 87
mlr_measures_classif.npv                     . 88
mlr_measures_classif.ppv                     . 89
mlr_measures_classif.prauc                   . 90
mlr_measures_classif.precision               . 91
mlr_measures_classif.recall                  . 92
mlr_measures_classif.sensitivity             . 94
mlr_measures_classif.specificity             . 95
mlr_measures_classif.tn                      . 96
mlr_measures_classif.tnr                     . 97
mlr_measures_classif.tp                      . 98
mlr_measures_classif.tpr                     .100
mlr_measures_debug                           .101
mlr_measures_elapsed_time                    .102
mlr_measures_oob_error                       .104
mlr_measures_regr.bias                      .105
mlr_measures_regr.ktau                       .106
mlr_measures_regr.mae                        .107
mlr_measures_regr.mapse                     .108
mlr_measures_regr.maxae                     .109
mlr_measures_regr.medae                     .110
mlr_measures_regr.median                    .111
mlr_measures_regr.mse                       .112
mlr_measures_regr.msle                      .113
mlr_measures_regr.pbias                    .114
mlr_measures_regr.rae                       .115
mlr_measures_regr.rmse                      .116
mlr_measures_regr.rmsle                     .117
mlr_measures_regr.rsse                      .118
mlr_measures_regr.rse                       .119
mlr_measures_regr.rsq                      .120
mlr_measures_regr.sae                       .121
mlr_measures_regr.smape                     .122
mlr_measures_regr.srho                      .123
mlr_measures_regr.sse                       .124
mlr_measures_selected_features             .125
mlr_resamplings                             .126
mlr_resamplings_bootstrap                   .127
mlr_resamplings_custom                      .129
mlr_resamplings_cv                           .131
mlr_resamplings_holdout                     .132
mlr_resamplings_insample                    .134
```
## R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mlr_resamplings_loo</td>
<td>135</td>
</tr>
<tr>
<td>mlr_resamplings_repeated_cv</td>
<td>137</td>
</tr>
<tr>
<td>mlr_resamplings_subsampling</td>
<td>139</td>
</tr>
<tr>
<td>mlr_sugar</td>
<td>141</td>
</tr>
<tr>
<td>mlr_tasks</td>
<td>142</td>
</tr>
<tr>
<td>mlr_tasks_boston_housing</td>
<td>144</td>
</tr>
<tr>
<td>mlr_tasks_breast_cancer</td>
<td>145</td>
</tr>
<tr>
<td>mlr_tasks_german_credit</td>
<td>146</td>
</tr>
<tr>
<td>mlr_tasks_iris</td>
<td>147</td>
</tr>
<tr>
<td>mlr_tasks_mtcars</td>
<td>148</td>
</tr>
<tr>
<td>mlr_tasks_penguins</td>
<td>149</td>
</tr>
<tr>
<td>mlr_tasks_pima</td>
<td>150</td>
</tr>
<tr>
<td>mlr_tasks_sonar</td>
<td>151</td>
</tr>
<tr>
<td>mlr_tasks_spam</td>
<td>152</td>
</tr>
<tr>
<td>mlr_tasks_wine</td>
<td>153</td>
</tr>
<tr>
<td>mlr_tasks_zoo</td>
<td>154</td>
</tr>
<tr>
<td>mlr_task_generators</td>
<td>155</td>
</tr>
<tr>
<td>mlr_task_generators_2dnormals</td>
<td>156</td>
</tr>
<tr>
<td>mlr_task_generators_cassini</td>
<td>157</td>
</tr>
<tr>
<td>mlr_task_generators_circle</td>
<td>158</td>
</tr>
<tr>
<td>mlr_task_generators_friedman1</td>
<td>160</td>
</tr>
<tr>
<td>mlr_task_generators_moons</td>
<td>161</td>
</tr>
<tr>
<td>mlr_task_generators_simplex</td>
<td>162</td>
</tr>
<tr>
<td>mlr_task_generators_smiley</td>
<td>164</td>
</tr>
<tr>
<td>mlr_task_generators_spirals</td>
<td>165</td>
</tr>
<tr>
<td>mlr_task_generators_xor</td>
<td>167</td>
</tr>
<tr>
<td>predict.Learner</td>
<td>168</td>
</tr>
<tr>
<td>Prediction</td>
<td>169</td>
</tr>
<tr>
<td>PredictionClassif</td>
<td>171</td>
</tr>
<tr>
<td>PredictionData</td>
<td>174</td>
</tr>
<tr>
<td>PredictionRegr</td>
<td>175</td>
</tr>
<tr>
<td>resample</td>
<td>176</td>
</tr>
<tr>
<td>ResampleResult</td>
<td>178</td>
</tr>
<tr>
<td>Resampling</td>
<td>182</td>
</tr>
<tr>
<td>set_threads</td>
<td>186</td>
</tr>
<tr>
<td>Task</td>
<td>187</td>
</tr>
<tr>
<td>TaskClassif</td>
<td>197</td>
</tr>
<tr>
<td>TaskGenerator</td>
<td>199</td>
</tr>
<tr>
<td>TaskRegr</td>
<td>201</td>
</tr>
</tbody>
</table>

### Index

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>204</td>
</tr>
</tbody>
</table>
mlr3-package

mlr3: Machine Learning in R - Next Generation

Description
Efficient, object-oriented programming on the building blocks of machine learning. Provides 'R6' objects for tasks, learners, resamplings, and measures. The package is geared towards scalability and larger datasets by supporting parallelization and out-of-memory data-backends like databases. While `mlr3` focuses on the core computational operations, add-on packages provide additional functionality.

Learn mlr3
- Use cases and examples gallery: https://mlr3gallery.mlr-org.com
- Cheat Sheets: https://cheatsheets.mlr-org.com

mlr3 extensions
- Preprocessing and machine learning pipelines: mlr3pipelines
- Analysis of benchmark experiments: mlr3benchmark
- More classification and regression tasks: mlr3data
- Connector to OpenML: mlr3oml
- Solid selection of good classification and regression learners: mlr3learners
- Even more learners: https://github.com/mlr-org/mlr3extralearners
- Tuning of hyperparameters: mlr3tuning
- Hyperband tuner: mlr3hyperband
- Visualizations for many mlr3 objects: mlr3viz
- Survival analysis and probabilistic regression: mlr3proba
- Cluster analysis: mlr3cluster
- Feature selection filters: mlr3filters
- Feature selection wrappers: mlr3fselect
- Interface to real (out-of-memory) data bases: mlr3db
- Performance measures as plain functions: mlr3measures

Suggested packages
- Parallelization framework: future
- Progress bars: progressr
- Encapsulated evaluation: evaluate, callr (external process)
**mlr3-package**

**Package Options**

- "mlr3.debug": If set to TRUE, parallelization via `future` is disabled to simplify debugging and provide more concise tracebacks. Note that results computed with debug mode enabled use a different seeding mechanism and are not reproducible.

- "mlr3.allow_utf8_names": If set to TRUE, checks on the feature names are relaxed, allowing non-ascii characters in column names. This is an experimental and temporal option to pave the way for text analysis, and will likely be removed in a future version of the package.

**Author(s)**

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

Authors:

- Bernd Bischl <bernd_bischl@gmx.net> (ORCID)
- Jakob Richter <jakob1richter@gmail.com> (ORCID)
- Patrick Schratz <patrick.schratz@gmail.com> (ORCID)
- Martin Binder <mlr.developer@mb706.com>

Other contributors:

- Giuseppe Casalicchio <giuseppe.casalicchio@stat.uni-muenchen.de> (ORCID) [contributor]
- Stefan Coors <mail@stefancoors.de> (ORCID) [contributor]
- Quay Au <quayau@gmail.com> (ORCID) [contributor]
- Marc Becker <marcbecker@posteo.de> (ORCID) [contributor]

**References**


**See Also**

Useful links:

- https://mlr3.mlr-org.com
- https://github.com/mlr-org/mlr3
### as_benchmark_result  
**Convert toBenchmarkResult**

**Description**

Convert object to a BenchmarkResult.

**Usage**

```r
as_benchmark_result(x, ...)
```

```r
## S3 method for class 'BenchmarkResult'
as_benchmark_result(x, ...)
```

```r
## S3 method for class 'ResampleResult'
as_benchmark_result(x, ...)
```

**Arguments**

- `x` (any)
  
  Object to convert.

- `...` (any)
  
  Additional arguments.

**Value**

(BenchmarkResult).

---

### as_data_backend.Matrix  
**Create a Data Backend**

**Description**

Wraps a DataBackend around data. mlr3 ships with methods for data.frame (converted to a DataBackendDataTable and Matrix from package Matrix (converted to a DataBackendMatrix). Additional methods are implemented in the package mlr3db, e.g. to connect to real DBMS like PostgreSQL (via dbplyr) or DuckDB (via DBI/duckdb).
Usage

## S3 method for class 'Matrix'
\[\text{as\_data\_backend}(\text{data}, \text{primary\_key} = \text{NULL}, \text{dense} = \text{NULL}, \ldots)\]

\[\text{as\_data\_backend}(\text{data}, \text{primary\_key} = \text{NULL}, \ldots)\]

## S3 method for class 'data.frame'
\[\text{as\_data\_backend}(\text{data}, \text{primary\_key} = \text{NULL}, \text{keep\_rownames} = \text{FALSE}, \ldots)\]

Arguments

- **data** *(data.frame())*
  The input \text{data.frame}. Converted to a \text{data.table::data.table()} automatically.

- **primary_key** *(character(1) | integer())*
  Name of the primary key column, or integer vector of row ids.

- **dense** *(data.frame())*
  Dense data.

- **...** *(any)*
  Additional arguments passed to the respective \text{DataBackend} method.

- **keep\_rownames** *(logical(1) | character(1))*
  If TRUE or a single string, keeps the row names of data as a new column. The column is named like the provided string, defaulting to \"..rownames\" for \text{keep\_rownames} == TRUE. Note that the created column will be used as a regular feature by the task unless you manually change the column role. Also see \text{data.table::as.data.table()}. 

Value

\text{DataBackend}.

See Also

Other \text{DataBackend}: \text{DataBackendDataTable, DataBackendMatrix, DataBackend}

Examples

\[
\begin{align*}
\# \text{create a new backend using the penguins data:} \\
\text{as\_data\_backend(palmerpenguins::penguins)}
\end{align*}
\]

\[
\begin{array}{ll}
\text{as\_learner} & \text{Convert to a Learner}
\end{array}
\]

Description

Convert object to a \text{Learner} or a list of \text{Learner}. 
Usage

as_learner(x, ...)

## S3 method for class 'Learner'
as_learner(x, clone = FALSE, ...)

as_learners(x, clone = FALSE, ...)

## S3 method for class 'list'
as_learners(x, clone = FALSE, ...)

## S3 method for class 'Learner'
as_learners(x, clone = FALSE, ...)

Arguments

x (any)
Object to convert.

... (any)
Additional arguments.

clone (logical(1))
If TRUE, ensures that the returned object is not the same as the input x.

Value

Learner.

as_measure

Convert to a Measure

Description

Convert object to a Measure or a list of Measure.

Usage

as_measure(x, ...)

## S3 method for class `\'NULL\'`
as_measure(x, task_type = NULL, clone = FALSE, ...)

## S3 method for class 'Measure'
as_measure(x, clone = FALSE, ...)

as_measures(x, ...)
as_prediction

## S3 method for class '\NULL\'
as_measures(x, task_type = NULL, clone = FALSE, ...)

## S3 method for class 'list'
as_measures(x, clone = FALSE, ...)

## S3 method for class 'Measure'
as_measures(x, clone = FALSE, ...)

Arguments

x (any)
Object to convert.

... (any)
Additional arguments.

task_type (character(1))
Used if x is NULL to construct a default measure for the respective task type. The
default measures are stored in mlr_reflections$default_measures.

clonelogical(1))
If TRUE, ensures that the returned object is not the same as the input x.

Value

Measure.

as_prediction Convert to a Prediction

Description

Convert object to a Prediction or a list of Prediction.

Usage

as_prediction(x, check = TRUE, ...)

## S3 method for class 'Prediction'
as_prediction(x, check = TRUE, ...)

## S3 method for class 'PredictionDataClassif'
as_prediction(x, check = TRUE, ...)

## S3 method for class 'PredictionDataRegr'
as_prediction(x, check = TRUE, ...)

as_predictions(x, predict_sets = "test", ...)
## S3 method for class 'list'
as_predictions(x, predict_sets = "test", ...)

### Arguments
- **x** (any): Object to convert.
- **check** (logical(1)): Perform argument checks and type conversions?
- **predict_sets** (character()): Prediction sets to operate on, used in aggregate() to extract the matching predict_sets from the ResampleResult. Multiple predict sets are calculated by the respective Learner during resample()/benchmark(). Must be a non-empty subset of {"train", "test", "validation"}. If multiple sets are provided, these are first combined to a single prediction object. Default is "test".

### Value
- **Prediction**.

---

**as_prediction_classif**  
*Convert to a Classification Prediction*

### Description
Convert object to a PredictionClassif.

### Usage

```r
as_prediction_classif(x, ...)  
```

**## S3 method for class 'PredictionClassif'**

```r
as_prediction_classif(x, ...)  
```

**## S3 method for class 'data.frame'**

```r
as_prediction_classif(x, ...)  
```

### Arguments
- **x** (any): Object to convert.
- **...** (any): Additional arguments.
\textbf{as\_prediction\_data}

\textbf{Value}

\texttt{PredictionClassif.}

\textbf{Examples}

\begin{verbatim}
# create a prediction object
task = tsk("penguins")
learner = lrn("classif.rpart", predict_type = "prob")
learner$train(task)
p = learner$predict(task)

# convert to a data.table
tab = as.data.table(p)

# convert back to a Prediction
as_prediction_classif(tab)

# split data.table into a list of data.tables
tabs = split(tab, tab$truth)

# convert back to list of predictions
preds = lapply(tabs, as_prediction_classif)

# calculate performance in each group
sapply(preds, function(p) p$score())
\end{verbatim}

---

\textbf{as\_prediction\_data \quad PredictionData}

\textbf{Description}

Convert object to a PredictionData or a list of PredictionData.

\textbf{Usage}

\begin{verbatim}
as_prediction_data(x, task, row_ids = task$row_ids, check = TRUE, ...)

## S3 method for class 'Prediction'
as_prediction_data(x, task, row_ids = task$row_ids, check = TRUE, ...)

## S3 method for class 'PredictionData'
as_prediction_data(x, task, row_ids = task$row_ids, check = TRUE, ...)

## S3 method for class 'list'
as_prediction_data(x, task, row_ids = task$row_ids, check = TRUE, ...)
\end{verbatim}
as_prediction_regr

Arguments

\( x \) (any)
Object to convert.

\( \text{task} \) (Task).

\( \text{row\_ids} \) (integer()).

\( \text{check} \) (logical(1))
Perform argument checks and type conversions?

\( \ldots \) (any)
Additional arguments.

Value

PredictionData.

---

as_prediction_regr  Convert to a Regression Prediction

Description

Convert object to a PredictionRegr.

Usage

as_prediction_regr(x, ...)

## S3 method for class 'PredictionRegr'
as_prediction_regr(x, ...)

## S3 method for class 'data.frame'
as_prediction_regr(x, ...)

Arguments

\( x \) (any)
Object to convert.

\( \ldots \) (any)
Additional arguments.

Value

PredictionRegr.
Examples

```r
# create a prediction object
task = tsk("mtcars")
learner = lrn("regr.rpart")
learner$train(task)
p = learner$predict(task)

# convert to a data.table
tab = as.data.table(p)

# convert back to a Prediction
as_prediction_regr(tab)

# split data.table into a list of data.tables
tabs = split(tab, cut(tab$truth, 3))

# convert back to list of predictions
preds = lapply(tabs, as_prediction_regr)

# calculate performance in each group
sapply(preds, function(p) p$score())
```

---

### as_resample_result

**Convert to ResampleResult**

**Description**

Convert object to a `ResampleResult`.

**Usage**

```r
as_resample_result(x, ...)
```

**Arguments**

- `x` (any): Object to convert.
- `...` (any): Currently not used.

**Value**

`(ResampleResult)`.
as_resampling  

Convert to a Resampling

Description

Convert object to a Resampling or a list of Resampling.

Usage

as_resampling(x, ...)

## S3 method for class 'Resampling'
as_resampling(x, clone = FALSE, ...)

as_resamplings(x, ...)

## S3 method for class 'list'
as_resamplings(x, clone = FALSE, ...)

## S3 method for class 'Resampling'
as_resamplings(x, clone = FALSE, ...)

Arguments

x (any)
Object to convert.

... (any)
Additional arguments.

clone (logical(1))
If TRUE, ensures that the returned object is not the same as the input x.

as_result_data  

Convert to ResultData

Description

This function allows to construct or convert to a ResultData object, the result container used by ResampleResult and BenchmarkResult. A ResampleResult or BenchmarkResult can be initialized with the returned object. Note that ResampleResults can be converted to a BenchmarkResult with as_benchmark_result() and multiple BenchmarkResults can be combined to a larger BenchmarkResult with the $combine() method of BenchmarkResult.
as_result_data

Usage

as_result_data(
  task,
  learners,
  resampling,
  iterations,
  predictions,
  learner_states = NULL,
  store_backends = TRUE
)

Arguments

task (Task).
learners (list of trained Learners).
resampling (Resampling).
iterations (integer()).
predictions (list of Predictions).
learner_states (list())
  Learner states. If not provided, the states of learners are automatically extracted.
store_backends (logical(1))
  If set to FALSE, the backends of the Tasks provided in data are removed.

Value

ResultData object which can be passed to the constructor of ResampleResult.

Examples

task = tsk("penguins")
learner = lrn("classif.rpart")
resampling = rsmp("cv", folds = 2)$instantiate(task)
iterations = seq_len(resampling$iters)

# manually train two learners.
# store learners and predictions
learners = list()
predictions = list()
for (i in iterations) {
  l = learner$clone(deep = TRUE)
  learners[[i]] = l$train(task, row_ids = resampling$train_set(i))
  predictions[[i]] = l$predict(task, row_ids = resampling$test_set(i))
}

rdata = as_result_data(task, learners, resampling, iterations, predictions)
ResampleResult$new(rdata)
as_task

Convert to a Task

Description

Convert object to a Task or a list of Task.

Usage

as_task(x, ...)

## S3 method for class 'Task'
as_task(x, clone = FALSE, ...)

as_tasks(x, ...)

## S3 method for class 'list'
as_tasks(x, clone = FALSE, ...)

## S3 method for class 'Task'
as_tasks(x, clone = FALSE, ...)

Arguments

x (any)
Object to convert.

... (any)
Additional arguments.

clone (logical(1))
If TRUE, ensures that the returned object is not the same as the input x.

as_task_classif

Convert to a Classification Task

Description

Convert object to a TaskClassif. This is a S3 generic, specialized for at least the following objects:

1. TaskClassif: ensure the identity
2. data.frame() and DataBackend: provides an alternative to the constructor of TaskClassif.
3. TaskRegr: Calls convert_task().
Usage

as_task_classif(x, ...)

## S3 method for class 'TaskClassif'
as_task_classif(x, clone = FALSE, ...)

## S3 method for class 'data.frame'
as_task_classif(
x,
target = NULL,
id = deparse(substitute(x)),
positive = NULL,
...
)

## S3 method for class 'DataBackend'
as_task_classif(
x,
target = NULL,
id = deparse(substitute(x)),
positive = NULL,
...
)

## S3 method for class 'TaskRegr'
as_task_classif(
x,
target = NULL,
drop_original_target = FALSE,
drop_levels = TRUE,
...
)

Arguments

x (any)
Object to convert.

... (any)
Additional arguments.

clone (logical(1))
If TRUE, ensures that the returned object is not the same as the input x.

target (character(1))
Name of the target column.

id (character(1))
Id for the new task. Defaults to the (deparsed and substituted) name of x.

positive (character(1))
Level of the positive class. See TaskClassif.
as_task_regr

```
drop_original_target (logical(1))
    If FALSE (default), the original target is added as a feature. Otherwise the original target is dropped.

drop_levels (logical(1))
    If TRUE (default), unused levels of the new target variable are dropped.
```

Value

TaskClassif.

Examples

```
as_task_classif(palmerpenguins::penguins, target = "species")
```

Description

Convert object to a TaskRegr. This is a S3 generic, specialized for at least the following objects:

1. TaskRegr: ensure the identity
2. data.frame() and DataBackend: provides an alternative to the constructor of TaskRegr.
3. TaskClassif: Calls convert_task().

Usage

```
as_task_regr(x, ...)
```

```
## S3 method for class 'TaskRegr'
as_task_regr(x, clone = FALSE, ...)

## S3 method for class 'data.frame'
as_task_regr(x, target, id = deparse(substitute(x)), ...)

## S3 method for class 'DataBackend'
as_task_regr(x, target, id = deparse(substitute(x)), ...)

## S3 method for class 'TaskClassif'
as_task_regr(
    x,
    target = NULL,
    drop_original_target = FALSE,
    drop_levels = TRUE,
    ...)
```
benchmark

Arguments

x (any)
Object to convert.

... (any)
Additional arguments.

clone (logical(1))
If TRUE, ensures that the returned object is not the same as the input x.

target (character(1))
Name of the target column.

id (character(1))
Id for the new task. Defaults to the (deparsed and substituted) name of x.

drop_original_target (logical(1))
If FALSE (default), the original target is added as a feature. Otherwise the original target is dropped.

drop_levels (logical(1))
If TRUE (default), unused levels of the new target variable are dropped.

Value

TaskRegr.

Examples

as_task_regr(datasets::mtcars, target = "mpg")

benchmark

Benchmark Multiple Learners on Multiple Tasks

Description

Runs a benchmark on arbitrary combinations of tasks (Task), learners (Learner), and resampling strategies (Resampling), possibly in parallel.

Usage

benchmark(design, store_models = FALSE, store_backends = TRUE)

Arguments

design (data.frame())
Data frame (or data.table::data.table()) with three columns: "task", "learner", and "resampling". Each row defines a resampling by providing a Task, Learner and an instantiated Resampling strategy. The helper function benchmark_grid() can assist in generating an exhaustive design (see examples) and instantiate the Resamplings per Task.
store_models (logical(1))
Store the fitted model in the resulting BenchmarkResult? Set to TRUE if you want to further analyse the models or want to extract information like variable importance.

store_backends (logical(1))
Keep the DataBackend of the Task in the BenchmarkResult? Set to TRUE if your performance measures require a Task, or to analyse results more conveniently. Set to FALSE to reduce the file size and memory footprint after serialization. The current default is TRUE, but this eventually will be changed in a future release.

Value
BenchmarkResult.

Parallelization
This function can be parallelized with the future package. One job is one resampling iteration, and all jobs are send to an apply function from future.apply in a single batch. To select a parallel backend, use future::plan().

Progress Bars
This function supports progress bars via the package progressr. Simply wrap the function in progressr::with_progress() to enable them. We recommend to use package progress as backend; enable with progressr::handlers("progress").

Logging
The mlr3 uses the lgr package for logging. lgr supports multiple log levels which can be queried with getOption("lgr.log_levels").

To suppress output and reduce verbosity, you can lower the log from the default level "info" to "warn":

lgr::get_logger("mlr3")$set_threshold("warn")

To get additional log output for debugging, increase the log level to "debug" or "trace":

lgr::get_logger("mlr3")$set_threshold("debug")

To log to a file or a data base, see the documentation of lgr::lgr-package.

Note
The fitted models are discarded after the predictions have been scored in order to reduce memory consumption. If you need access to the models for later analysis, set store_models to TRUE.
Examples

```r
# benchmarking with benchmark_grid()
tasks = lapply(c("penguins", "sonar"), tsk)
learners = lapply(c("classif.featureless", "classif.rpart"), lrn)
resamplings = rsmp("cv", folds = 3)

design = benchmark_grid(tasks, learners, resamplings)
print(design)

set.seed(123)
bmr = benchmark(design)

## Data of all resamplings
head(as.data.table(bmr))

## Aggregated performance values
aggr = bmr$aggregate()
print(aggr)

## Extract predictions of first resampling result
rr = aggr$resample_result[[1]]
as.data.table(rr$prediction())

# Benchmarking with a custom design:
# - fit classif.featureless on penguins with a 3-fold CV
# - fit classif.rpart on sonar using a holdout
tasks = list(tsk("penguins"), tsk("sonar"))
learners = list(lrn("classif.featureless"), lrn("classif.rpart"))
resamplings = list(rsmp("cv", folds = 3), rsmp("holdout"))

design = data.table::data.table(
  task = tasks,
  learner = learners,
  resampling = resamplings
)

## Instantiate resamplings
design$resampling = Map(
  function(task, resampling) resampling$clone()$instantiate(task),
  task = design$task, resampling = design$resampling
)

## Run benchmark
bmr = benchmark(design)
print(bmr)

## Get the training set of the 2nd iteration of the featureless learner on penguins
rr = bmr$aggregate()[learner_id == "classif.featureless"]$resample_result[[1]]
rr$resampling$train_set(2)
```
**BenchmarkResult**

**Description**

This is the result container object returned by `benchmark()`. A `BenchmarkResult` consists of the data row-bound data of multiple `ResampleResults`, which can easily be re-constructed. `BenchmarkResults` can be visualized via `mlr3viz`’s `autoplot()` function.

For statistical analysis of benchmark results and more advanced plots, see `mlr3benchmark`.

**S3 Methods**

- `as.data.table(rr,...,reassemble_learners = TRUE,convert_predictions = TRUE,predict_sets = "test")`
  `BenchmarkResult` -> `data.table::data.table()`
  Returns a tabular view of the internal data.
- `c(...)`
  `(BenchmarkResult, ...)` -> `BenchmarkResult`
  Combines multiple objects convertible to `BenchmarkResult` into a new `BenchmarkResult`.

**Public fields**

- `data` (ResultData)
  Internal data storage object of type `ResultData`. We discourage users to directly work with this field. Use `as.table.table(BenchmarkResult)` instead.

**Active bindings**

- `task_type` (character(1))
  Task type of objects in the `BenchmarkResult`. All stored objects (Task, Learner, Prediction) in a single `BenchmarkResult` are required to have the same task type, e.g., "classif" or "regr". This is NA for empty `BenchmarkResults`.
- `tasks` (`data.table::data.table()`) Table of included Tasks with three columns:
  - "task_hash" (character(1)),
  - "task_id" (character(1)), and
  - "task" (Task).
- `learners` (`data.table::data.table()`) Table of included Learners with three columns:
  - "learner_hash" (character(1)),
  - "learner_id" (character(1)), and
  - "learner" (Learner).

Note that it is not feasible to access learned models via this field, as the training task would be ambiguous. For this reason the returned learner are reseted before they are returned. Instead, select a row from the table returned by `$score()`.
`BenchmarkResult`

resamplings (```data.table::data.table()```
  Table of included `Resamplings` with three columns:
  • "resampling_hash" (character(1)),
  • "resampling_id" (character(1)), and
  • "resampling" (``Resampling``).
resample_results (```data.table::data.table()```
  Returns a table with three columns:
  • uhash (character()).
  • resample_result (``ResampleResult``).

n_resample_results (``integer(1)``)
  Returns the total number of stored `ResampleResults`.

uhashes (character())
  Set of (unique) hashes of all included `ResampleResults`.

Methods

Public methods:
• `BenchmarkResult$new()`
• `BenchmarkResult$help()`
• `BenchmarkResult$format()`
• `BenchmarkResult$print()`
• `BenchmarkResult$combine()`
• `BenchmarkResult$score()`
• `BenchmarkResult$aggregate()`
• `BenchmarkResult$filter()`
• `BenchmarkResult$resample_result()`
• `BenchmarkResult$clone()`

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

Usage:
`BenchmarkResult$new(data = NULL)`

Arguments:

data (``ResultData``)
  An object of type ResultData, either extracted from another `ResampleResult`, another
  `BenchmarkResult`, or manually constructed with `as_result_data()`.

Method `help()`: Opens the help page for this object.

Usage:
`BenchmarkResult$help()`

Method `format()`: Helper for print outputs.

Usage:
`BenchmarkResult$format()`
Method print(): Printer.

Usage:
BenchmarkResult$print()

Method combine(): Fuses a second BenchmarkResult into itself, mutating the BenchmarkResult in-place. If the second BenchmarkResult bmr is NULL, simply returns self. Note that you can alternatively use the combine function c() which calls this method internally.

Usage:
BenchmarkResult$combine(bmr)

Arguments:

bmr (BenchmarkResult)
A second BenchmarkResult object.

Returns: Returns the object itself, but modified by reference. You need to explicitly $clone() the object beforehand if you want to keep the object in its previous state.

Method score(): Returns a table with one row for each resampling iteration, including all involved objects: Task, Learner, Resampling, iteration number (integer(1)), and Prediction. If ids is set to TRUE, character column of extracted ids are added to the table for convenient filtering: "task_id", "learner_id", and "resampling_id".

Additionally calculates the provided performance measures and binds the performance scores as extra columns. These columns are named using the id of the respective Measure.

Usage:
BenchmarkResult$score(
  measures = NULL,
  ids = TRUE,
  conditions = FALSE,
  predict_sets = "test"
)

Arguments:

measures (Measure | list of Measure)
Measure(s) to calculate.

ids (logical(1))
Adds object ids ("task_id", "learner_id", "resampling_id") as extra character columns to the returned table.

conditions (logical(1))
Adds condition messages ("warnings", "errors") as extra list columns of character vectors to the returned table.

predict_sets (character())
Prediction sets to operate on, used in aggregate() to extract the matching predict_sets from the ResampleResult. Multiple predict sets are calculated by the respective Learner during resample()/benchmark(). Must be a non-empty subset of ("train", "test", "validation"). If multiple sets are provided, these are first combined to a single prediction object. Default is "test".

Returns: data.table::data.table().
**Method** aggregate(): Returns a result table where resampling iterations are combined into ResampleResults. A column with the aggregated performance score is added for each Measure, named with the id of the respective measure.

For convenience, different flags can be set to extract more information from the returned ResampleResult:

**Usage:**
BenchmarkResult$aggregate(
  measures = NULL,
  ids = TRUE,
  uhashes = FALSE,
  params = FALSE,
  conditions = FALSE
)

**Arguments:**
- measures (Measure | list of Measure)
  Measure(s) to calculate.
- ids (logical(1))
  Adds object ids ("task_id", "learner_id", "resampling_id") as extra character columns for convenient subsetting.
- uhashes (logical(1))
  Adds the uhash values of the ResampleResult as extra character column "uhash".
- params (logical(1))
  Adds the hyperparameter values as extra list column "params". You can unnest them with mlr3misc::unnest().
- conditions (logical(1))
  Adds the number of resampling iterations with at least one warning as extra integer column "warnings", and the number of resampling iterations with errors as extra integer column "errors".

**Returns:** data.table::data.table().

**Method** filter(): Subsets the benchmark result. If task_ids is not NULL, keeps all tasks with provided task ids and discards all others tasks. Same procedure for learner_ids and resampling_ids.

**Usage:**
BenchmarkResult$filter(
  task_ids = NULL,
  task_hashes = NULL,
  learner_ids = NULL,
  learner_hashes = NULL,
  resampling_ids = NULL,
  resampling_hashes = NULL
)

**Arguments:**
- task_ids (character())
  Ids of Tasks to keep.
task_hashes (character())
   Hashes of Tasks to keep.
learner_ids (character())
   Ids of Learners to keep.
learner_hashes (character())
   Hashes of Learners to keep.
resampling_ids (character())
   Ids of Resamplings to keep.
resampling_hashes (character())
   Hashes of Resamplings to keep.

Returns: Returns the object itself, but modified by reference. You need to explicitly $clone()$ the object beforehand if you want to keep the object in its previous state.

Method resample_result(): Retrieve the i-th ResampleResult, by position or by unique hash uhash. i and uhash are mutually exclusive.

Usage:
BenchmarkResult$resample_result(i = NULL, uhash = NULL)

Arguments:
   i (integer(1))
      The iteration value to filter for.
   uhash (logical(1))
      The uhash value to filter for.

Returns: ResampleResult.

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

Usage:
BenchmarkResult$clone(deep = FALSE)

Arguments:
   deep Whether to make a deep clone.

Note
All stored objects are accessed by reference. Do not modify any extracted object without cloning it first.

Examples
set.seed(123)
learners = list(
   lrn("classif.featureless", predict_type = "prob"),
   lrn("classif.rpart", predict_type = "prob")
)
design = benchmark_grid(
   tasks = list(tsk("sonar"), tsk("spam")),
   learners = learners,
)
— benchmark_grid —

```
resamplings = rsmp("cv", folds = 3)
)
print(design)

bmr = benchmark(design)
print(bmr)

bmr$tasks
bmr$learners

# first 5 resampling iterations
head(as.data.table(bmr, measures = c("classif.acc", "classif.auc")), 5)

# aggregate results
bmr$aggregate()

# aggregate results with hyperparameters as separate columns
mlr3misc::unnest(bmr$aggregate(params = TRUE), "params")

# extract resample result for classif.rpart
rr = bmr$aggregate()[learner_id == "classif.rpart", resample_result][[1]]
print(rr)

# access the confusion matrix of the first resampling iteration
rr$predictions()[[1]]$confusion

# reduce to subset with task id "sonar"
bmr$filter(task_ids = "sonar")
print(bmr)
```

### benchmark_grid

**Generate a Benchmark Grid Design**

**Description**

Takes a lists of Task, a list of Learner and a list of Resampling to generate a design in an `expand.grid()` fashion (a.k.a. cross join or Cartesian product).

Resampling strategies are not allowed to be instantiated when passing the argument, and instead will be instantiated per task internally. The only exception to this rule applies if all tasks have exactly the same number of rows, and the resamplings are all instantiated for such tasks.

**Usage**

`benchmark_grid(tasks, learners, resamplings)`

**Arguments**

- `tasks` (list of `Task`).
- `learners` (list of `Learner`).
- `resamplings` (list of `Resampling`).
convert_task

Convert a Task from One Type to Another

Description

The task’s target is replaced by a different column from the data.

Usage

convert_task(
  intask, 
  target = NULL, 
  new_type = NULL, 
  drop_original_target = FALSE, 
  drop_levels = TRUE 
)

Arguments

intask (Task)
A Task to be converted.

target (character(1))
New target to be set, must be a column in the intask data. If NULL, no new target is set, and task is converted as-is.

new_type (character(1))
The new task type. Must be in mlr_reflections$task_types. If NULL (default), a new task with the same task_type is created.

drop_original_target
(logical(1))
If FALSE (default), the original target is added as a feature. Otherwise the original target is dropped.

drop_levels (logical(1))
If TRUE (default), unused levels of the new target variable are dropped.

Value

Task of requested type.
Description

This is the abstract base class for data backends.

Data backends provide a layer of abstraction for various data storage systems. It is not recommended to work directly with the DataBackend. Instead, all data access is handled transparently via the Task.

This package comes with two implementations for backends:

- DataBackendDataTable which stores the data as data.table::data.table().
- DataBackendMatrix which stores the data as sparse Matrix::sparseMatrix().

To connect to out-of-memory database management systems such as SQL servers, see the extension package mlr3db.

The required set of fields and methods to implement a custom DataBackend is listed in the respective sections (see DataBackendDataTable or DataBackendMatrix for exemplary implementations of the interface).

Public fields

primary_key (character(1))
  Column name of the primary key column of unique integer row ids.

data_formats (character())
  Set of supported formats, e.g. "data.table" or "Matrix".

Active bindings

hash (character(1))
  Hash (unique identifier) for this object.

Methods

Public methods:

- DataBackend$new()
- DataBackend$format()
- DataBackend$print()

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

Note: This object is typically constructed via a derived classes, e.g. DataBackendDataTable or DataBackendMatrix, or via the S3 method as_data_backend().

Usage:
DataBackend$new(data, primary_key, data_formats = "data.table")

Arguments:
data (any)
The format of the input data depends on the specialization. E.g., DataBackendDataTable expects a data.table::data.table() and DataBackendMatrix expects a Matrix::Matrix() from Matrix.

primary_key (character(1))
Each DataBackend needs a way to address rows, which is done via a column of unique integer values, referenced here by primary_key. The use of this variable may differ between backends.

data_formats (character())
Set of supported data formats which can be processed during $train()$ and $predict()$, e.g. "data.table".

Method format(): Helper for print outputs.
Usage:
DataBackend$format()

Method print(): Printer.
Usage:
DataBackend$print()

See Also
Extension Packages: mlr3db
Other DataBackend: DataBackendDataTable, DataBackendMatrix, as_data_backend.Matrix()

Examples

data = data.table::data.table(id = 1:5, x = runif(5),
y = sample(letters[1:3], 5, replace = TRUE))

b = DataBackendDataTable$new(data, primary_key = "id")

print(b)
b$head(2)
b$data(rows = 1:2, cols = "x")
b$distinct(rows = b$rownames, "y")
b$missings(rows = b$rownames, cols = names(data))
Public fields

compact_seq logical(1)
If TRUE, row ids are a natural sequence from 1 to nrow(data) (determined internally). In this case, row lookup uses faster positional indices instead of equi joins.

Active bindings

rownames (integer())
Returns vector of all distinct row identifiers, i.e. the contents of the primary key column.

colnames (character())
Returns vector of all column names, including the primary key column.

nrow (integer(1))
Number of rows (observations).

ncol (integer(1))
Number of columns (variables), including the primary key column.

Methods

Public methods:

- `DataBackendDataTable$new()`
- `DataBackendDataTable$data()`
- `DataBackendDataTable$head()`
- `DataBackendDataTable$distinct()`
- `DataBackendDataTable$missings()`

Method `new()`: Creates a new instance of this R6 class.
Note that `DataBackendDataTable` does not copy the input data, while `as_data_backend()` calls `data.table::copy()`. `as_data_backend()` also takes care about casting to a `data.table()` and adds a primary key column if necessary.

Usage:
`DataBackendDataTable$new(data, primary_key)`

Arguments:

data (data.table::data.table())
The input `data.table()`.
primary_key (character(1)|integer())
Name of the primary key column, or integer vector of row ids.

Method `data()`: Returns a slice of the data in the specified format. Currently, the only supported formats are "data.table" and "Matrix". The rows must be addressed as vector of primary key values, columns must be referred to via column names. Queries for rows with no matching row id and queries for columns with no matching column name are silently ignored. Rows are guaranteed to be returned in the same order as `rows`, columns may be returned in an arbitrary order. Duplicated row ids result in duplicated rows, duplicated column names lead to an exception.

Usage:
`DataBackendDataTable$data(rows, cols, data_format = "data.table")`
Arguments:
rows integer()
Row indices.
cols character()
Column names.
data_format character(1)
Desired data format, e.g. "data.table" or "Matrix".

Method head(): Retrieve the first n rows.
Usage:
DataBackendDataTable$head(n = 6L)
Arguments:
n (integer(1))
Number of rows.
Returns: data.table::data.table() of the first n rows.

Method distinct(): Returns a named list of vectors of distinct values for each column specified. If na_rm is TRUE, missing values are removed from the returned vectors of distinct values. Non-existing rows and columns are silently ignored.
Usage:
DataBackendDataTable$distinct(rows, cols, na_rm = TRUE)
Arguments:
rows integer()
Row indices.
cols character()
Column names.
na_rm logical(1)
Whether to remove NAs or not.
Returns: Named list() of distinct values.

Method missings(): Returns the number of missing values per column in the specified slice of data. Non-existing rows and columns are silently ignored.
Usage:
DataBackendDataTable$missings(rows, cols)
Arguments:
rows integer()
Row indices.
cols character()
Column names.
Returns: Total of missing values per column (named numeric()).

See Also
Other DataBackend: DataBackendMatrix, DataBackend, as_data_backend.Matrix()
Examples

```r
data = as.data.table(palmerpenguins::penguins)
data$id = seq_len(nrow(palmerpenguins::penguins))
b = DataBackendDataTable$new(data = data, primary_key = "id")
print(b)
b$head()
b$data(rows = 100:101, cols = "species")

b$nrow
head(b$rownames)

b$ncol
b$colnames

# alternative construction
as_data_backend(palmerpenguins::penguins)
```

DataBackendMatrix  DataBackend for Matrix

Description

DataBackend for Matrix. Data is split into a (numerical) sparse part and an optional dense part. These parts are automatically merged to a sparse format during $data(). Note that merging both parts potentially comes with a data loss, as all dense columns are converted to numeric columns.

Super class

mlr3::DataBackend -> DataBackendMatrix

Active bindings

- **rownames (integer())**
  
  Returns vector of all distinct row identifiers, i.e. the contents of the primary key column.

- **colnames (character())**
  
  Returns vector of all column names, including the primary key column.

- **nrow (integer(1))**
  
  Number of rows (observations).

- **ncol (integer(1))**
  
  Number of columns (variables), including the primary key column.

Methods

Public methods:

- DataBackendMatrix$new()
- DataBackendMatrix$data()
• `DataBackendMatrix$head()`
• `DataBackendMatrix$distinct()`
• `DataBackendMatrix$missings()`

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

**Usage:**
`DataBackendMatrix$new(data, dense = NULL, primary_key = NULL)`

**Arguments:**
- `data` `Matrix::Matrix()`
  - The input `Matrix::Matrix()`
- `dense` `data.frame()`
  - Dense data, converted to `data.table::data.table()`
- `primary_key` `character(1) | integer()`
  - Name of the primary key column, or integer vector of row ids.

**Method data():** Returns a slice of the data in the specified format. Currently, the only supported formats are "data.table" and "Matrix". The rows must be addressed as vector of primary key values, columns must be referred to via column names. Queries for rows with no matching row id and queries for columns with no matching column name are silently ignored. Rows are guaranteed to be returned in the same order as `rows`, columns may be returned in an arbitrary order. Duplicated row ids result in duplicated rows, duplicated column names lead to an exception.

**Usage:**
`DataBackendMatrix$data(rows, cols, data_format = "data.table")`

**Arguments:**
- `rows` `integer()`
  - Row indices.
- `cols` `character()`
  - Column names.
- `data_format` `character(1)`
  - Desired data format, e.g. "data.table" or "Matrix".

**Method head():** Retrieve the first $n$ rows.

**Usage:**
`DataBackendMatrix$head(n = 6L)`

**Arguments:**
- `n` `integer(1)`
  - Number of rows.

**Returns:** `data.table::data.table()` of the first $n$ rows.

**Method distinct():** Returns a named list of vectors of distinct values for each column specified. If `na_rm` is `TRUE`, missing values are removed from the returned vectors of distinct values. Non-existing rows and columns are silently ignored.

**Usage:**
`DataBackendMatrix$distinct(rows, cols, na_rm = TRUE)`

**Arguments:**
**Method** missings(): Returns the number of missing values per column in the specified slice of data. Non-existing rows and columns are silently ignored.

**Usage:**
```
DataBackendMatrix$missings(rows, cols)
```

**Arguments:**
- `rows` integer(): Row indices.
- `cols` character(): Column names.

**Returns:** Total of missing values per column (named numeric()).

**See Also**
Other DataBackend: `DataBackendDataTable`, `DataBackend`, `as_data_backend.Matrix()`

**Examples**
```
requireNamespace("Matrix")
data = Matrix::Matrix(sample(0:1, 20, replace = TRUE), ncol = 2)
colnames(data) = c("x1", "x2")
dense = data.frame(
  ..row_id = 1:10,
  num = runif(10),
  fact = factor(sample(c("a", "b"), 10, replace = TRUE), levels = c("a", "b"))
)

b = as_data_backend(data, dense = dense, primary_key = ".row_id")
b$head()
b$data(1:3, b$colnames, data_format = "Matrix")
b$data(1:3, b$colnames, data_format = "data.table")
```

---

**default_measures**

**Get a Default Measure**

**Description**

Gets the default measures using the information in `mlr_reflections$default_measures`:
- "classif.ce" for classification ("classif").
- "regr.mse" for regression ("regr").
- Add-on package may register additional default measures for their own task types.
Usage

default_measures(task_type)

Arguments

task_type (character(1))
Get the default measure for the task type task_type, e.g., "classif" or "regr". If task_type is NULL, an empty list is returned.

Value

list of Measure.

Examples

default_measures("classif")
default_measures("regr")

---

Learner Learner Class

Description

This is the abstract base class for learner objects like LearnerClassif and LearnerRegr.

Learners are build around the three following key parts:

- Methods $train() and $predict() which call internal methods (either public method $train_internal/$predict_internal() (deprecated) or private methods $.train/$.predict()).
- A paradox::ParamSet which stores meta-information about available hyperparameters, and also stores hyperparameter settings.
- Meta-information about the requirements and capabilities of the learner.
- The fitted model stored in field $model, available after calling $train().

Predefined learners are stored in the dictionary mlr_learners, e.g. classif.rpart or regr.rpart.

More classification and regression learners are implemented in the add-on package mlr3learners. Learners for survival analysis (or more general, for probabilistic regression) can be found in mlr3proba. Unsupervised cluster algorithms are implemented in mlr3cluster. The dictionary mlr_learners gets automatically populated with the new learners as soon as the respective packages are loaded.

More (experimental) learners can be found in the GitHub repository: https://github.com/mlr-org/mlr3extralearners. A guide on how to extend mlr3 with custom learners can be found in the mlr3book.
Optional Extractors

Specific learner implementations are free to implement additional getters to ease the access of certain parts of the model in the inherited subclasses.

For the following operations, extractors are standardized:

- `importance(...)`: Returns the feature importance score as numeric vector. The higher the score, the more important the variable. The returned vector is named with feature names and sorted in decreasing order. Note that the model might omit features it has not used at all. The learner must be tagged with property "importance". To filter variables using the importance scores, see package `mlr3filters`.

- `selected_features(...)`: Returns a subset of selected features as character(). The learner must be tagged with property "selected_features".

- `oob_error(...)`: Returns the out-of-bag error of the model as numeric(1). The learner must be tagged with property "oob_error".

Setting Hyperparameters

All information about hyperparameters is stored in the slot `param_set` which is a `paradox::ParamSet`. The printer gives an overview about the ids of available hyperparameters, their storage type, lower and upper bounds, possible levels (for factors), default values and assigned values. To set hyperparameters, assign a named list to the subslot `values`:

```r
lrn = lrn("classif.rpart")
lrn$param_set$values = list(minsplit = 3, cp = 0.01)
```

Note that this operation replaces all previously set hyperparameter values. If you only intend to change one specific hyperparameter value and leave the others as-is, you can use the helper function `mlr3misc::insert_named()`:

```r
lrn$param_set$values = mlr3misc::insert_named(lrn$param_set$values, list(cp = 0.001))
```

If the learner has additional hyperparameters which are not encoded in the `ParamSet`, you can easily extend the learner. Here, we add a factor hyperparameter with id "foo" and possible levels "a" and "b":

```r
lrn$param_set$add(paradox::ParamFct$new("foo", levels = c("a", "b")))
```

Public fields

- `id` (character(1))
  Identifier of the object. Used in tables, plot and text output.

- `state` (NULL | named list())
  Current (internal) state of the learner. Contains all information gathered during `train()` and `predict()`. It is not recommended to access elements from `state` directly. This is an internal data structure which may change in the future.

- `task_type` (character(1))
  Task type, e.g. "classif" or "regr".
  For a complete list of possible task types (depending on the loaded packages), see `mlr_reflections$task_types$type`
predict_types (character())
Stores the possible predict types the learner is capable of. A complete list of candidate predict
types, grouped by task type, is stored in `mlr_reflections$learner_predict_types`.

feature_types (character())
Stores the feature types the learner can handle, e.g. "logical", "numeric", or "factor". A
complete list of candidate feature types, grouped by task type, is stored in `mlr_reflections$task_feature_types`.

properties (character())
Stores a set of properties/capabilities the learner has. A complete list of candidate properties,
grouped by task type, is stored in `mlr_reflections$learner_properties`.

data_formats (character())
Supported data format, e.g. "data.table" or "Matrix".

packages (character(1))
Set of required packages. These packages are loaded, but not attached.

predict_sets (character())
During `resample()/benchmark()`, a Learner can predict on multiple sets. Per default, a
learner only predicts observations in the test set (predict_sets == "test"). To change
this behaviour, set predict_sets to a non-empty subset of {"train", "test", "validation"}. Each set yields a separate Prediction object. Those be combined via getters in ResampleRe-
sult/BenchmarkResult, or Measures can be altered to operate on specific subsets of the calcu-
lated prediction sets.

timeout (named numeric(2))
Timeout for the learner's train and predict steps, in seconds. This works differently for differ-
ent encapsulation methods, see `mlr3misc::encapsulate()`. Default is c(train = Inf, predict
= Inf).

fallback (Learner)
Learner which is fitted to impute predictions in case that either the model fitting or the pre-
diction of the top learner is not successful. Requires you to enable encapsulation, otherwise
errors are not caught and the execution is terminated before the fallback learner kicks in.

man (character(1))
String in the format [pkg]::[topic] pointing to a manual page for this object. Defaults to NA, but can be set by child classes.

Active bindings

model (any)
The fitted model. Only available after $train() has been called.

timings (named numeric(2))
Elapsed time in seconds for the steps "train" and "predict". Measured via `mlr3misc::encapsulate()`.

log (data.table::data.table())
Returns the output (including warning and errors) as table with columns
• "stage" ("train" or "predict"),
• "class" ("output", "warning", or "error"), and
• "msg" (character()).

warnings (character())
Logged warnings as vector.
errors (character())
  Logged errors as vector.
hash (character(1))
  Hash (unique identifier) for this object.
phash (character(1))
  Hash (unique identifier) for this partial object, excluding some components which are varied
  systematically during tuning (parameter values) or feature selection (feature names).
predict_type (character(1))
  Stores the currently active predict type, e.g. "response". Must be an element of $predict_types.
param_set (paradox::ParamSet)
  Set of hyperparameters.
encapsulate (named character())
  Controls how to execute the code in internal train and predict methods. Must be a named character vector with names "train" and "predict". Possible values are "none", "evaluate" (requires package evaluate) and "callr" (requires package callr). See mlr3misc::encapsulate() for more details.

Methods

Public methods:
  • Learner$new()
  • Learner$format()
  • Learner$print()
  • Learner$help()
  • Learner$train()
  • Learner$predict()
  • Learner$predict_newdata()
  • Learner$reset()
  • Learner$clone()

Method new(): Creates a new instance of this R6 class.
Note that this object is typically constructed via a derived classes, e.g. LearnerClassif or LearnerRegr.

Usage:
Learner$new(
  id,
  task_type,
  param_set = ParamSet$new(),
  predict_types = character(),
  feature_types = character(),
  properties = character(),
  data_formats = "data.table",
  packages = character(),
  man = NA_character_ 
)
Arguments:

id (character(1))
   Identifier for the new instance.

task_type (character(1))
   Type of task, e.g. "regr" or "classif". Must be an element of mlr_reflections$task_types$type.

param_set (paradox::ParamSet)
   Set of hyperparameters.

predict_types (character())
   Supported predict types. Must be a subset of mlr_reflections$learner_predict_types.

feature_types (character())
   Feature types the learner operates on. Must be a subset of mlr_reflections$task_feature_types.

properties (character())
   Set of properties of the Learner. Must be a subset of mlr_reflections$learner_properties.
   The following properties are currently standardized and understood by learners in mlr3:
   • "missings": The learner can handle missing values in the data.
   • "weights": The learner supports observation weights.
   • "importance": The learner supports extraction of importance scores, i.e. comes with an
     $importance() extractor function (see section on optional extractors in Learner).
   • "selected_features": The learner supports extraction of the set of selected features,
     i.e. comes with a $selected_features() extractor function (see section on optional extrac-
     tors in Learner).
   • "oob_error": The learner supports extraction of estimated out of bag error, i.e. comes
     with a oob_error() extractor function (see section on optional extractors in Learner).

data_formats (character())
   Set of supported data formats which can be processed during $train() and $predict(), e.g.
   "data.table".

packages (character())
   Set of required packages. A warning is signaled by the constructor if at least one of the pack-
   ages is not installed, but loaded (not attached) later on-demand via requireNamespace().

man (character(1))
   String in the format [pkg]::[topic] pointing to a manual page for this object. The referenced
   help package can be opened via method $help().

Method format(): Helper for print outputs.
   Usage:
   Learner$format()

Method print(): Printer.
   Usage:
   Learner$print()

Arguments:
   ... (ignored).

Method help(): Opens the corresponding help page referenced by field $man.
   Usage:
Learner$help()

**Method train():** Train the learner on a set of observations of the provided task. Mutates the learner by reference, i.e. stores the model alongside other information in field $state.

*Usage:*
Learner$train(task, row_ids = NULL)

*Arguments:*
- task (Task).
- row_ids (integer())
  Vector of training indices.

*Returns:* Returns the object itself, but modified by reference. You need to explicitly $clone() the object beforehand if you want to keep the object in its previous state.

**Method predict():** Uses the information stored during $train() in $state to create a new Prediction for a set of observations of the provided task.

*Usage:*
Learner$predict(task, row_ids = NULL)

*Arguments:*
- task (Task).
- row_ids (integer())
  Vector of test indices.

*Returns:* Prediction.

**Method predict_newdata():** Uses the model fitted during $train() to create a new Prediction based on the new data in newdata. Object task is the task used during $train() and required for conversion of newdata. If the learner’s $train() method has been called, there is a (size reduced) version of the training task stored in the learner. If the learner has been fitted via resample() or benchmark(), you need to pass the corresponding task stored in the ResampleResult or BenchmarkResult, respectively.

*Usage:*
Learner$predict_newdata(newdata, task = NULL)

*Arguments:*
- newdata (data.frame())
  New data to predict on. Row ids are automatically set to 1:nrow(newdata).
- task (Task).

*Returns:* Prediction.

**Method reset():** Reset the learner, i.e. un-train by resetting the state.

*Usage:*
Learner$reset()

*Returns:* Returns the object itself, but modified by reference. You need to explicitly $clone() the object beforehand if you want to keep the object in its previous state.

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

Arguments:
depth Whether to make a deep clone.

See Also

Other Learner: LearnerClassif, LearnerRegr, mlr_learners_classif.debug, mlr_learners_classif.featureless, mlr_learners_classif.rpart, mlr_learners_regr.featureless, mlr_learners_regr.rpart, mlr_learners

LearnerClassif  Classification Learner

Description

This Learner specializes Learner for classification problems:

- task_type is set to "classif".
- Creates Predictions of class PredictionClassif.
- Possible values for predict_types are:
  - "response": Predicts a class label for each observation in the test set.
  - "prob": Predicts the posterior probability for each class for each observation in the test set.
- Additional learner properties include:
  - "twoclass": The learner works on binary classification problems.
  - "multiclass": The learner works on multiclass classification problems.

Predefined learners can be found in the dictionary mlr_learners. Essential classification learners can be found in this dictionary after loading mlr3learners. Additional learners are implement in the Github package https://github.com/mlr-org/mlr3extralearners.

Super class

mlr3::Learner -> LearnerClassif

Methods

Public methods:
- LearnerClassif$new()
- LearnerClassif$clone()

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

Usage:
LearnerClassif$new(
    id,
    param_set = ParamSet$new(),
    predict_types = "response",
    feature_types = character(),
    properties = character(),
    data_formats = "data.table",
    packages = character(),
    man = NA_character_
)

Arguments:

id (character(1))
  Identifier for the new instance.

param_set (paradox::ParamSet)
  Set of hyperparameters.

predict_types (character())
  Supported predict types. Must be a subset of mlr_reflections$learner_predict_types.

feature_types (character())
  Feature types the learner operates on. Must be a subset of mlr_reflections$task_feature_types.

properties (character())
  Set of properties of the Learner. Must be a subset of mlr_reflections$learner_properties.
  The following properties are currently standardized and understood by learners in mlr3:
  • "missings": The learner can handle missing values in the data.
  • "weights": The learner supports observation weights.
  • "importance": The learner supports extraction of importance scores, i.e. comes with an $importance() extractor function (see section on optional extractors in Learner).
  • "selected_features": The learner supports extraction of the set of selected features, i.e. comes with a $selected_features() extractor function (see section on optional extractors in Learner).
  • "oob_error": The learner supports extraction of estimated out of bag error, i.e. comes with a oob_error() extractor function (see section on optional extractors in Learner).

data_formats (character())
  Set of supported data formats which can be processed during $train() and $predict(), e.g. "data.table".

packages (character())
  Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via requireNamespace().

man (character(1))
  String in the format [pkg]:[topic] pointing to a manual page for this object. The referenced help package can be opened via method $help().

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

Usage:
LearnerClassif$clone(deep = FALSE)

Arguments:

deep  Whether to make a deep clone.
See Also


Examples

```r
# get all classification learners from mlr_learners:
lrns = mlr_learners$mget(mlr_learners$keys("^classif"))
names(lrns)

# get a specific learner from mlr_learners:
lrn = lrn("classif.rpart")
print(lrn)

# train the learner:
task = tsk("penguins")
lrn$train(task, 1:200)

# predict on new observations:
lrn$predict(task, 201:344)$confusion
```

LearnerRegr  
Regression Learner

Description

This Learner specializes Learner for regression problems:

- `task_type` is set to "regr".
- Creates Predictions of class PredictionRegr.
- Possible values for `predict_types` are:
  - "response": Predicts a numeric response for each observation in the test set.
  - "se": Predicts the standard error for each value of response for each observation in the test set.
  - "distr": Probability distribution as distr6::VectorDistribution object (requires package distr6).

Predefined learners can be found in the dictionary mlr_learners. Essential regression learners can be found in this dictionary after loading mlr3learners. Additional learners are implement in the Github package https://github.com/mlr-org/mlr3extralearners.

Super class

mlr3::Learner -> LearnerRegr
Methods

Public methods:

- `LearnerRegr$new()`
- `LearnerRegr$clone()`

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

Usage:

```r
LearnerRegr$new(
  id,
  param_set = ParamSet$new(),
  predict_types = "response",
  feature_types = character(),
  properties = character(),
  data_formats = "data.table",
  packages = character(),
  man = NA_character_
)
```

Arguments:

- `id` (character(1))
  Identifier for the new instance.
- `param_set` (paradox::ParamSet)
  Set of hyperparameters.
- `predict_types` (character())
  Supported predict types. Must be a subset of `mlr_reflections$learner_predict_types`.
- `feature_types` (character())
  Feature types the learner operates on. Must be a subset of `mlr_reflections$task_feature_types`.
- `properties` (character())
  Set of properties of the Learner. Must be a subset of `mlr_reflections$learner_properties`.
  The following properties are currently standardized and understood by learners in mlr3:
  - "missings": The learner can handle missing values in the data.
  - "weights": The learner supports observation weights.
  - "importance": The learner supports extraction of importance scores, i.e. comes with an $importance() extractor function (see section on optional extractors in Learner).
  - "selected_features": The learner supports extraction of the set of selected features, i.e. comes with a $selected_features() extractor function (see section on optional extractors in Learner).
  - "oob_error": The learner supports extraction of estimated out of bag error, i.e. comes with a oob_error() extractor function (see section on optional extractors in Learner).
- `data_formats` (character())
  Set of supported data formats which can be processed during $train() and $predict(), e.g. "data.table".
- `packages` (character())
  Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via requireNamespace().
String in the format [pkg]::[topic] pointing to a manual page for this object. The referenced help package can be opened via method $help().

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

*Usage:*
LearnerRegr$clone(deep = FALSE)

*Arguments:*
depth Whether to make a deep clone.

**See Also**
Other Learner: LearnerClassif, Learner, mlr_learners_classif.debug, mlr_learners_classif.featureless, mlr_learners_classif.rpart, mlr_learners_regr.featureless, mlr_learners_regr.rpart, mlr_learners

**Examples**

```r
# get all regression learners from mlr_learners:
lrns = mlr_learners$mget(mlr_learners$keys("regr"))
names(lrns)

# get a specific learner from mlr_learners:
mlr_learners$get("regr.rpart")
lrn("classif.featureless")
```

---

**Measure**

<table>
<thead>
<tr>
<th>Measure Class</th>
</tr>
</thead>
</table>

**Description**

This is the abstract base class for measures like MeasureClassif and MeasureRegr.

Measures are classes around tailored around two functions:

1. A function $score() which quantifies the performance by comparing true and predicted response.
2. A function $aggregator() which combines multiple performance scores returned by calculate to a single numeric value.

In addition to these two functions, meta-information about the performance measure is stored.

Predefined measures are stored in the dictionary mlr_measures, e.g. classif.auc or time_train. Many of the measures in mlr3 are implemented in mlr3measures as ordinary functions.

A guide on how to extend mlr3 with custom measures can be found in the mlr3book.
**Public fields**

- **id** (character(1))
  Identifier of the object. Used in tables, plot and text output.

- **task_type** (character(1))
  Task type, e.g. "classif" or "regr".
  For a complete list of possible task types (depending on the loaded packages), see `mlr_reflections$task_types$type`.

- **predict_type** (character(1))
  Required predict type of the Learner.

- **predict_sets** (character())
  During `resample()`/`benchmark()`, a Learner can predict on multiple sets. Per default, a learner only predicts observations in the test set (`predict_sets == "test"`). To change this behaviour, set `predict_sets` to a non-empty subset of {"train", "test", "validation"}. Each set yields a separate `Prediction` object. Those be combined via getters in `ResampleResult/BenchmarkResult`, or `Measures` can be altered to operate on specific subsets of the calculated prediction sets.

- **average** (character(1))
  Method for aggregation:
  - "micro": All predictions from multiple resampling iterations are first combined into a single `Prediction` object. Next, the scoring function of the measure is applied on this combined object, yielding a single numeric score.
  - "macro": The scoring function is applied on the `Prediction` object of each resampling iterations, each yielding a single numeric score. Next, the scores are combined with the `aggregator` function to a single numerical score.

- **aggregator** (function())
  Function to aggregate scores computed on different resampling iterations.

- **task_properties** (character())
  Required properties of the Task.

- **range** (numeric(2))
  Lower and upper bound of possible performance scores.

- **properties** (character())
  Properties of this measure.

- **minimize** (logical(1))
  If TRUE, good predictions correspond to small values of performance scores.

- **packages** (character(1))
  Set of required packages. These packages are loaded, but not attached.

- **man** (character(1))
  String in the format [pkg]::[topic] pointing to a manual page for this object. Defaults to `NA`, but can be set by child classes.

**Active bindings**

- **hash** (character(1))
  Hash (unique identifier) for this object.
Methods

Public methods:

• `Measure$new()`
• `Measure$format()`
• `Measure$print()`
• `Measure/help()`
• `Measure$score()`
• `Measure$aggregate()`

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

Note that this object is typically constructed via a derived classes, e.g. `MeasureClassif` or `MeasureRegr`.

Usage:

```r
Measure$new(
  id,
  task_type = NA,
  range = c(-Inf, Inf),
  minimize = NA,
  average = "macro",
  aggregator = NULL,
  properties = character(),
  predict_type = "response",
  predict_sets = "test",
  task_properties = character(),
  packages = character(),
  man = NA_character_
)
```

Arguments:

- `id` (character(1))
  Identifier for the new instance.
- `task_type` (character(1))
  Type of task, e.g. "regr" or "classif". Must be an element of `mlr_reflections$task_types$type`.
- `range` (numeric(2))
  Feasible range for this measure as `c(lower_bound, upper_bound)`. Both bounds may be infinite.
- `minimize` (logical(1))
  Set to TRUE if good predictions correspond to small values, and to FALSE if good predictions correspond to large values. If set to NA (default), tuning this measure is not possible.
- `average` (character(1))
  How to average multiple `Predictions` from a `ResampleResult`.
  The default, "macro", calculates the individual performances scores for each `Prediction` and then uses the function defined in `Saggregator` to average them to a single number.
  If set to "micro", the individual `Prediction` objects are first combined into a single new `Prediction` object which is then used to assess the performance. The function in `Saggregator` is not used in this case.
aggregator (function(x))
  Function to aggregate individual performance scores x where x is a numeric vector. If NULL, defaults to mean().

properties (character())
  Properties of the measure. Must be a subset of mlr_reflections$measure_properties. Supported by mlr3:
  • "requires_task" (requires the complete Task),
  • "requires_learner" (requires the trained Learner),
  • "requires_train_set" (requires the training indices from the Resampling), and
  • "na_score" (the measure is expected to occasionally return NA or NaN).

predict_type (character(1))
  Required predict type of the Learner. Possible values are stored in mlr_reflections$learner_predict_types.

predict_sets (character())
  Prediction sets to operate on, used in aggregate() to extract the matching predict_sets from the ResampleResult. Multiple predict sets are calculated by the respective Learner during resample()/benchmark(). Must be a non-empty subset of {"train", "test", "validation"}. If multiple sets are provided, these are first combined to a single prediction object. Default is "test".

task_properties (character())
  Required task properties, see Task.

packages (character())
  Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via requireNamespace().

man (character(1))
  String in the format [pkg]::[topic] pointing to a manual page for this object. The referenced help package can be opened via method $help().

Method format(): Helper for print outputs.

  Usage:
  Measure$format()

Method print(): Printer.

  Usage:
  Measure$print()

  Arguments:
  ... (ignored).

Method help(): Opens the corresponding help page referenced by field $man.

  Usage:
  Measure$help()

Method score(): Takes a Prediction (or a list of Prediction objects named with valid predict_sets) and calculates a numeric score. If the measure if flagged with the properties "requires_task", "requires_learner", "requires_model" or "requires_train_set", you must additionally pass the respective Task, the (trained) Learner or the training set indices. This is handled internally during resample()/benchmark().
MeasureClassif

Usage:
Measure$score(prediction, task = NULL, learner = NULL, train_set = NULL)

Arguments:
prediction (Prediction | named list of Prediction).
task (Task).
learner (Learner).
train_set (integer()).
Returns: numeric(1).

Method aggregate(): Aggregates multiple performance scores into a single score using the aggregator function of the measure. Operates on the Predictions of ResampleResult with matching predict_sets.

Usage:
Measure$aggregate(rr)

Arguments:
rr ResampleResult.
Returns: numeric(1).

See Also
Other Measure: MeasureClassif, MeasureRegr, mlr_measures_classif.costs, mlr_measures_debug, mlr_measures_elapsed_time, mlr_measures_oob_error, mlr_measures_selected_features, mlr_measures

---

MeasureClassif  Classification Measure

Description
This measure specializes Measure for classification problems:

- task_type is set to "classif".
- Possible values for predict_type are "response" and "prob".

Predefined measures can be found in the dictionary mlr_measures.

Super class
mlr3::Measure -> MeasureClassif
Methods

Public methods:
- `MeasureClassif$new()`

Method `new()`: Creates a new instance of this 

Usage:

```r
MeasureClassif$new(
  id,
  range,
  minimize = NA,
  average = "macro",
  aggregator = NULL,
  properties = character(),
  predict_type = "response",
  predict_sets = "test",
  task_properties = character(),
  packages = character(),
  man = NA_character_
)
```

Arguments:

- `id` (character(1))
  Identifier for the new instance.

- `range` (numeric(2))
  Feasible range for this measure as `c(lower_bound, upper_bound)`. Both bounds may be infinite.

- `minimize` (logical(1))
  Set to `TRUE` if good predictions correspond to small values, and to `FALSE` if good predictions correspond to large values. If set to `NA` (default), tuning this measure is not possible.

- `average` (character(1))
  How to average multiple Predictions from a ResampleResult.
  The default, "macro", calculates the individual performances scores for each Prediction and then uses the function defined in $aggregator to average them to a single number.
  If set to "micro", the individual Prediction objects are first combined into a single new Prediction object which is then used to assess the performance. The function in $aggregator is not used in this case.

- `aggregator` (function(x))
  Function to aggregate individual performance scores x where x is a numeric vector. If NULL, defaults to `mean()`.

- `properties` (character())
  Properties of the measure. Must be a subset of `mlr_reflections$measure_properties`. Supported by mlr3:
  - "requires_task" (requires the complete Task),
  - "requires_learner" (requires the trained Learner),
  - "requires_train_set" (requires the training indices from the Resampling), and
  - "na_score" (the measure is expected to occasionally return NA or NaN).
predict_type (character(1))
required predict type of the Learner. Possible values are stored in mlr_reflections$learner_predict_types.

predict_sets (character())
Prediction sets to operate on, used in aggregate() to extract the matching predict_sets from the ResampleResult. Multiple predict sets are calculated by the respective Learner during resample()/benchmark(). Must be a non-empty subset of \{"train", "test", "validation"\}. If multiple sets are provided, these are first combined to a single prediction object. Default is "test".

task_properties (character())
required task properties, see Task.

packages (character())
Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via requireNamespace().

man (character(1))
String in the format [pkg]:[topic] pointing to a manual page for this object. The referenced help package can be opened via method $help().

See Also
Default classification measures: classif.ce
Other Measure: MeasureRegr, Measure, mlr_measures_classif.costs, mlr_measures_debug, mlr_measures_elapsed_time, mlr_measures_oob_error, mlr_measures_selected_features, mlr_measures

MeasureRegr Regression Measure

Description
This measure specializes Measure for regression problems:
• task_type is set to "regr".
• Possible values for predict_type are "response", "se" and "distr".

Predefined measures can be found in the dictionary mlr_measures.

Super class
mlr3::Measure -> MeasureRegr

Methods
Public methods:
• MeasureRegr$new()

Method new(): Creates a new instance of this R6 class.
Usage:
MeasureRegr$new(
  id,
  range, 
  minimize = NA,
  average = "macro",
  aggregator = NULL,
  properties = character(),
  predict_type = "response",
  predict_sets = "test",
  task_properties = character(),
  packages = character(),
  man = NA_character_,
)

Arguments:
id (character(1))
  Identifier for the new instance.
range (numeric(2))
  Feasible range for this measure as c(lower_bound, upper_bound). Both bounds may be infinite.
minimize (logical(1))
  Set to TRUE if good predictions correspond to small values, and to FALSE if good predictions correspond to large values. If set to NA (default), tuning this measure is not possible.
average (character(1))
  How to average multiple Predictions from a ResampleResult.
  The default, "macro", calculates the individual performances scores for each Prediction and then uses the function defined in $aggregator to average them to a single number.
  If set to "micro", the individual Prediction objects are first combined into a single new Prediction object which is then used to assess the performance. The function in $aggregator is not used in this case.
aggregator (function(x))
  Function to aggregate individual performance scores x where x is a numeric vector. If NULL, defaults to mean().
properties (character())
  Properties of the measure. Must be a subset of mlr_reflections$measure_properties. Supported by mlr3:
  • "requires_task" (requires the complete Task),
  • "requires_learner" (requires the trained Learner),
  • "requires_train_set" (requires the training indices from the Resampling), and
  • "na_score" (the measure is expected to occasionally return NA or NaN).
predict_type (character(1))
  Required predict type of the Learner. Possible values are stored in mlr_reflections$learner_predict_types.
predict_sets (character())
  Prediction sets to operate on, used in aggregate() to extract the matching predict_sets from the ResampleResult. Multiple predict sets are calculated by the respective Learner
during `resample()`/`benchmark()`. Must be a non-empty subset of \{"train", "test", "validation"\}. If multiple sets are provided, these are first combined to a single prediction object. Default is "test".

task_properties (character())
Required task properties, see Task.

packages (character())
Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via `requireNamespace()`.

man (character(1))
String in the format [pkg][::][topic] pointing to a manual page for this object. The referenced help package can be opened via method $help().

See Also

Default regression measures: `regr.mse`

Other Measure: `MeasureClassif`, `Measure`, `mlr_measures_classif.costs`, `mlr_measures_debug`, `mlr_measures_elapsed_time`, `mlr_measures_oob_error`, `mlr_measures_selected_features`, `mlr_measures`

---

**mlr_learners**

*Dictionary of Learners*

**Description**

A simple `mlr3misc::Dictionary` storing objects of class Learner. Each learner has an associated help page, see `mlr_learners_[id]`.

This dictionary can get populated with additional learners by add-on packages. For an opinionated set of solid classification and regression learners, install and load the `mlr3learners` package. More learners are connected via `https://github.com/mlr-org/mlr3extralearners`.

For a more convenient way to retrieve and construct learners, see `lrn()`/`lrns()`.

**Format**

*R6::R6Class* object inheriting from `mlr3misc::Dictionary`.

**Methods**

See `mlr3misc::Dictionary`.

**S3 methods**

- `as.data.table(dict)`
  
  `mlr3misc::Dictionary` -> `data.table::data.table()`

  Returns a `data.table::data.table()` with fields "key", "feature_types", "packages", "properties" and "predict_types" as columns.
mlr_learners_classif.debug

See Also
Sugar functions: lrn(), lrs()
Extension Packages: mlr3learners
Other Dictionary: mlr_measures, mlr_resamplings, mlr_task_generators, mlr_tasks
Other Learner: LearnerClassif, LearnerRegr, Learner, mlr_learners_classif.debug, mlr_learners_classif.featureless, mlr_learners_classif.rpart, mlr_learners_regr.featureless, mlr_learners_regr.rpart

Examples
as.data.table(mlr_learners)
mlr_learners$get("classif.featureless")
lrn("classif.rpart")

mlr_learners_classif.debug

Classification Learner for Debugging

Description
A simple LearnerClassif used primarily in the unit tests and for debugging purposes. If no hyperparameter is set, it simply constantly predicts a randomly selected label. The following hyperparameters trigger the following actions:

message_train: Probability to output a message during train.
message_predict: Probability to output a message during predict.
warning_train: Probability to signal a warning during train.
warning_predict: Probability to signal a warning during predict.
error_train: Probability to raise an exception during train.
error_predict: Probability to raise an exception during predict.
segfault_train: Probability to provoke a segfault during train.
segfault_predict: Probability to provoke a segfault during predict.
predict_missing Ratio of predictions which will be NA.
save_tasks: Saves input task in model slot during training and prediction.
threads: Number of threads to use. Has no effect.
x: Numeric tuning parameter. Has no effect.

Note that segfaults may not be triggered on your operating system. Also note that if they work, they will tear down your R session immediately!

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

mlr_learners$get("classif.featureless")
lrn("classif.featureless")
Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “character”, “factor”, “ordered”
- Required Packages: -

Parameters

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Range</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>message_train</td>
<td>numeric</td>
<td>0</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>message_predict</td>
<td>numeric</td>
<td>0</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>warning_train</td>
<td>numeric</td>
<td>0</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>warning_predict</td>
<td>numeric</td>
<td>0</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>error_train</td>
<td>numeric</td>
<td>0</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>error_predict</td>
<td>numeric</td>
<td>0</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>segfault_train</td>
<td>numeric</td>
<td>0</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>segfault_predict</td>
<td>numeric</td>
<td>0</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>predict_missing</td>
<td>numeric</td>
<td>0</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>save_tasks</td>
<td>logical</td>
<td>FALSE</td>
<td>(−∞, ∞)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>threads</td>
<td>integer</td>
<td>-</td>
<td>[1, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>x</td>
<td>numeric</td>
<td>-</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
</tbody>
</table>

Super classes

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

Methods

Public methods:

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

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

Usage:

\texttt{LearnerClassifDebug$new()}

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

Usage:

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

Arguments:

deepe Whether to make a deep clone.
See Also

Dictionary of Learners: mlr_learners

as.data.table(mlr_learners) for a complete table of all (also dynamically created) Learner implementations.

Other Learner: LearnerClassif, LearnerRegr, Learner, mlr_learners_classif.featureless, mlr_learners_classif.rpart, mlr_learners_regr.featureless, mlr_learners_regr.rpart, mlr_learners

Examples

learner = lrn("classif.debug")
learner$param_set$values = list(message_train = 1, save_tasks = TRUE)

# this should signal a message
task = tsk("penguins")
learner$train(task)
learner$predict(task)

# task_train and task_predict are the input tasks for train() and predict()
names(learner$model)

mlr_learners_classif.featureless

Featureless Classification Learner

Description

A simple LearnerClassif which only analyses the labels during train, ignoring all features. Hyperparameter method determines the mode of operation during prediction:

- **mode**: Predicts the most frequent label. If there are two or more labels tied, randomly selects one per prediction.
- **sample**: Randomly predict a label uniformly.
- **weighted.sample**: Randomly predict a label, with probability estimated from the training distribution.

Dictionary

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

mlr_learners$get("classif.featureless")
lrn("classif.featureless")
Meta Information

- Task type: “classif”
- Predict Types: “response”, “prob”
- Required Packages: -

Parameters

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Range</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>method</td>
<td>character</td>
<td>mode</td>
<td>(−∞, ∞)</td>
<td>mode, sample, weighted.sample</td>
</tr>
</tbody>
</table>

Super classes

`mlr3::Learner` -> `mlr3::LearnerClassif` -> `LearnerClassifFeatureless`

Methods

**Public methods:**
- `LearnerClassifFeatureless$new()`
- `LearnerClassifFeatureless$importance()`
- `LearnerClassifFeatureless$selected_features()`
- `LearnerClassifFeatureless$clone()`

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

*Usage:*

```r
LearnerClassifFeatureless$new()
```

**Method importance():** All features have a score of 0 for this learner.

*Usage:*

```r
LearnerClassifFeatureless$importance()
```

*Returns:* Named numeric().

**Method selected_features():** Selected features are always the empty set for this learner.

*Usage:*

```r
LearnerClassifFeatureless$selected_features()
```

*Returns:* character(0).

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

*Usage:*

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

*Arguments:*

deep: Whether to make a deep clone.
**See Also**

Dictionary of Learners: `mlr_learners`

as.data.table(mlr_learners) for a complete table of all (also dynamically created) Learner implementations.

Other Learner: `LearnerClassif, LearnerRegr, Learner, mlr_learners_classif.debug, mlr_learners_classif.rpart, mlr_learners_regr.featureless, mlr_learners_regr.rpart, mlr_learners`

---

`mlr_learners_classif.rpart`

*Classification Tree Learner*

**Description**

A LearnerClassif for a classification tree implemented in `rpart::rpart()` in package `rpart`. Parameter `xval` is set to 0 in order to save some computation time. Parameter `model` has been renamed to `keep_model`.

**Dictionary**

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

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

**Meta Information**

- Task type: “classif”
- Predict Types: “response”, “prob”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: `rpart`

**Parameters**

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Range</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>minsplit</td>
<td>integer</td>
<td>20</td>
<td>([1, \infty))</td>
<td>-</td>
</tr>
<tr>
<td>minbucket</td>
<td>integer</td>
<td>-</td>
<td>([1, \infty))</td>
<td>-</td>
</tr>
<tr>
<td>cp</td>
<td>numeric</td>
<td>0.01</td>
<td>([0, 1])</td>
<td>-</td>
</tr>
<tr>
<td>maxcompete</td>
<td>integer</td>
<td>4</td>
<td>([0, \infty))</td>
<td>-</td>
</tr>
<tr>
<td>maxsurrogate</td>
<td>integer</td>
<td>5</td>
<td>([0, \infty))</td>
<td>-</td>
</tr>
<tr>
<td>maxdepth</td>
<td>integer</td>
<td>30</td>
<td>([1, 30])</td>
<td>-</td>
</tr>
<tr>
<td>usesurrogate</td>
<td>integer</td>
<td>2</td>
<td>([0, 2])</td>
<td>-</td>
</tr>
<tr>
<td>surrogatestyle</td>
<td>integer</td>
<td>0</td>
<td>([0, 1])</td>
<td>-</td>
</tr>
<tr>
<td>xval</td>
<td>integer</td>
<td>10</td>
<td>([0, \infty))</td>
<td>-</td>
</tr>
<tr>
<td>keep_model</td>
<td>logical</td>
<td>FALSE</td>
<td>((-\infty, \infty))</td>
<td>TRUE, FALSE</td>
</tr>
</tbody>
</table>
Super classes

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

Methods

Public methods:

- \texttt{LearnerClassifRpart$new()}
- \texttt{LearnerClassifRpart$importance()}
- \texttt{LearnerClassifRpart$selected\_features()}
- \texttt{LearnerClassifRpart$clone()}

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

\textit{Usage:}
\texttt{LearnerClassifRpart$new()}

\textbf{Method} \texttt{importance()}: The importance scores are extracted from the model slot \texttt{variable.importance}.

\textit{Usage:}
\texttt{LearnerClassifRpart$importance()}

\textit{Returns:} Named \texttt{numeric}.

\textbf{Method} \texttt{selected\_features()}: Selected features are extracted from the model slot \texttt{frame$var}.

\textit{Usage:}
\texttt{LearnerClassifRpart$selected\_features()}

\textit{Returns:} \texttt{character}.

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

\textit{Usage:}
\texttt{LearnerClassifRpart$clone(deep = FALSE)}

\textit{Arguments:}
- deep Whether to make a deep clone.

References


See Also

- \texttt{Dictionary of Learners: mlr\_learners}
- \texttt{as.data.table(mlr_learners)} for a complete table of all (also dynamically created) \texttt{Learner} implementations.
- Other Learner: \texttt{LearnerClassif,LearnerRegr,Learner,mlr_learners_classif.debug,mlr_learners_classif.featureless,mlr_learners_regr.featureless,mlr_learners_regr.rpart,mlr_learners}
Featureless Regression Learner

Description

A simple LearnerRegr which only analyses the response during train, ignoring all features. If hyperparameter robust is FALSE (default), constantly predicts mean(y) as response and sd(y) as standard error. If robust is TRUE, median() and mad() are used instead of mean() and sd(), respectively.

Dictionary

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

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

Meta Information

- Task type: "regr"
- Predict Types: "response", "se"
- Feature Types: "logical", "integer", "numeric", "character", "factor", "ordered", "POSIXct"
- Required Packages: 'stats'

Parameters

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Range</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>robust</td>
<td>logical</td>
<td>TRUE</td>
<td>(−∞, ∞)</td>
<td>TRUE, FALSE</td>
</tr>
</tbody>
</table>

Super classes

`mlr3::Learner` -> `mlr3::LearnerRegr` -> LearnerRegrFeatureless

Methods

- LearnerRegrFeatureless$new()
- LearnerRegrFeatureless$importance()
- LearnerRegrFeatureless$selected_features()
- LearnerRegrFeatureless$clone()`
Method new(): Creates a new instance of this R6 class.
  Usage:
  LearnerRegrFeatureless$new()

Method importance(): All features have a score of 0 for this learner.
  Usage:
  LearnerRegrFeatureless$importance()
  Returns: Named numeric().

Method selected_features(): Selected features are always the empty set for this learner.
  Usage:
  LearnerRegrFeatureless$selected_features()
  Returns: character(0).

Method clone(): The objects of this class are cloneable with this method.
  Usage:
  LearnerRegrFeatureless$clone(deep = FALSE)
  Arguments:
  deep Whether to make a deep clone.

See Also

Dictionary of Learners: mlr_learners
as.data.table(mlr_learners) for a complete table of all (also dynamically created) Learner implementations.
Other Learner: LearnerClassif, LearnerRegr, Learner, mlr_learners_classif.debug, mlr_learners_classif.featureless, mlr_learners_classif.rpart, mlr_learners_regr.rpart, mlr_learners

mlr_learners_regr.rpart

Regression Tree Learner

Description

Parameter xval is set to 0 in order to save some computation time. Parameter model has been renamed to keep_model.

Dictionary

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

mlr_learners$get("regr.rpart")
lrn("regr.rpart")
Meta Information

- Task type: “regr”
- Predict Types: “response”
- Feature Types: “logical”, “integer”, “numeric”, “factor”, “ordered”
- Required Packages: rpart

Parameters

<table>
<thead>
<tr>
<th>Id</th>
<th>Type</th>
<th>Default</th>
<th>Range</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>minsplit</td>
<td>integer</td>
<td>20</td>
<td>[1, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>minbucket</td>
<td>integer</td>
<td>-</td>
<td>[1, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>cp</td>
<td>numeric</td>
<td>0.01</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>maxcompete</td>
<td>integer</td>
<td>4</td>
<td>[0, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>maxsurrogate</td>
<td>integer</td>
<td>5</td>
<td>[0, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>maxdepth</td>
<td>integer</td>
<td>30</td>
<td>[1, 30]</td>
<td>-</td>
</tr>
<tr>
<td>usesurrogate</td>
<td>integer</td>
<td>2</td>
<td>[0, 2]</td>
<td>-</td>
</tr>
<tr>
<td>surrogatestyle</td>
<td>integer</td>
<td>0</td>
<td>[0, 1]</td>
<td>-</td>
</tr>
<tr>
<td>xval</td>
<td>integer</td>
<td>10</td>
<td>[0, ∞)</td>
<td>-</td>
</tr>
<tr>
<td>keep_model</td>
<td>logical</td>
<td>FALSE</td>
<td>(−∞, ∞)</td>
<td>TRUE, FALSE</td>
</tr>
</tbody>
</table>

Super classes

mlr3::Learner -> mlr3::LearnerRegr -> LearnerRegrRpart

Methods

Public methods:

- LearnerRegrRpart$new()
- LearnerRegrRpart$importance()
- LearnerRegrRpart$selected_features()
- LearnerRegrRpart$clone()

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

Usage:
LearnerRegrRpart$new()

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

Usage:
LearnerRegrRpart$importance()

Returns: Named numeric().

Method selected_features(): Selected features are extracted from the model slot frame$var.
Usage:
LearnerRegrRpart$selected_features()

Returns: character().

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

Usage:
LearnerRegrRpart$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

References

See Also
Dictionary of Learners: mlr_learners
as.data.table(mlr_learners) for a complete table of all (also dynamically created) Learner implementations.

Other Learner: LearnerClassif, LearnerRegr, Learner, mlr_learners_classif.debug, mlr_learners_classif.featureless, mlr_learners_classif.rpart, mlr_learners_regr.featureless, mlr_learners

mlr_measures

Dictionary of Performance Measures

Description
A simple mlr3misc::Dictionary storing objects of class Measure. Each measure has an associated help page, see mlr_measures_[id].

This dictionary can get populated with additional measures by add-on packages. E.g., mlr3proba adds survival measures and mlr3cluster adds cluster analysis measures.

For a more convenient way to retrieve and construct measures, see msr()/msrs().

Format
R6::R6Class object inheriting from mlr3misc::Dictionary.

Methods
See mlr3misc::Dictionary.
S3 methods

- as.data.table(dict)
  
  \texttt{mlr3misc::Dictionary -> data.table::data.table()}
  
  Returns a \texttt{data.table::data.table()} with fields "key", "task_type", "predict_type", and "packages" as columns.

See Also

Sugar functions: \texttt{msr()}, \texttt{msrs()}

Implementation of most measures: \texttt{mlr3measures}

Other Dictionary: \texttt{mlr_learners, mlr_resamplings, mlr_task_generators, mlr_tasks}

Other Measure: \texttt{MeasureClassif, MeasureRegr, Measure, mlr_measures_classif.costs, mlr_measures_debug, mlr_measures_elapsed_time, mlr_measures_oob_error, mlr_measures_selected_features}

Examples

- \texttt{as.data.table(mlr_measures)}
- \texttt{mlr_measures$get("classif.ce")}
- \texttt{msr("regr.mse")}

```
mlr_measures_classif.acc

\textit{Classification Accuracy}
```

Description

Classification measure defined as

\[
\frac{1}{n} \sum_{i=1}^{n} (t_i = r_i).
\]

Dictionary

This \texttt{Measure} can be instantiated via the \texttt{dictionary mlr_measures} or with the associated sugar function \texttt{msr()}:

- \texttt{mlr_measures$get("acc")}
- \texttt{msr("acc")}

Meta Information

- Type: "classif"
- Range: [0, 1]
- Minimize: FALSE
- Required prediction: response
The score function calls `mlr3measures::acc()` from package `mlr3measures`.
If the measure is undefined for the input, `NaN` is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures`
as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


Other multiclass classification measures: `mlr_measures_classif.bacc`, `mlr_measures_classif.ce`, `mlr_measures_classif.costs`, `mlr_measures_classif.logloss`, `mlr_measures_classif.mbrier`

---

### Description

Computes the area under the Receiver Operator Characteristic (ROC) curve. The AUC can be interpreted as the probability that a randomly chosen positive observation has a higher predicted probability than a randomly chosen negative observation.

### Dictionary

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
classif.auc
msr("auc")
```

### Meta Information

- Type: "binary"
- Range: [0, 1]
- Minimize: FALSE
- Required prediction: prob
Note

The score function calls `mlr3measures::auc()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures` as.data.table(`mlr_measures`) for a complete table of all (also dynamically created) Measure implementations.


---

**mlr_measures_classif.bacc**

*Balanced Accuracy*

**Description**

Computes the weighted balanced accuracy, suitable for imbalanced data sets. It is defined analogously to the definition in sklearn.

First, the sample weights \( w \) are normalized per class:

\[
\hat{w}_i = \frac{w_i}{\sum_j 1(y_j = y_i)w_i}.
\]

The balanced accuracy is calculated as

\[
\frac{1}{\sum_i w_i} \sum_i 1(r_i = t_i)\hat{w}_i.
\]
Dictionary

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("bacc")
msr("bacc")
```

Meta Information

- Type: "classif"
- Range: [0, 1]
- Minimize: FALSE
- Required prediction: response

Note

The score function calls `mlr3measures::bacc()` from package `mlr3measures`. If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures`

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


Other multiclass classification measures: `mlr_measures_classif.acc, mlr_measures_classif.ce, mlr_measures_classif.costs, mlr_measures_classif.logloss, mlr_measures_classif.mbrier`
Description

Brier score for binary classification problems defined as

\[ \frac{1}{n} \sum_{i=1}^{n} (I_i - p_i)^2. \]

\( I_i \) is 1 if observation \( i \) belongs to the positive class, and 0 otherwise.

Note that this (more common) definition of the Brier score is equivalent to the original definition of the multi-class Brier score (see `mbrier()`) divided by 2.

Dictionary

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("bbrier")
msr("bbrier")
```

Meta Information

- Type: "binary"
- Range: [0, 1]
- Minimize: TRUE
- Required prediction: prob

Note

The score function calls `mlr3measures::bbrier()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures`
as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


mlr_measures_classif.ce

Classification Error

Description

Classification measure defined as

$$\frac{1}{n} \sum_{i=1}^{n} (t_i \neq r_i).$$

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("ce")
msr("ce")
```

Meta Information

- Type: "classif"
- Range: [0,1]
- Minimize: TRUE
- Required prediction: response

Note

The score function calls mlr3measures::ce() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

Cost-sensitive Classification Measure

Description
Uses a cost matrix to create a classification measure. True labels must be arranged in columns, predicted labels must be arranged in rows. The cost matrix is stored as slot $costs.

For calculation of the score, the confusion matrix is multiplied element-wise with the cost matrix. The costs are then summed up (and potentially divided by the number of observations if normalize is set to TRUE).

This measure requires the Task during scoring to ensure that the rows and columns of the cost matrix are in the same order as in the confusion matrix.

Dictionary
This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

mlr_measures$get("classif.costs")
msr("classif.costs")

Meta Information
- Type: "classif"
- Range: $[0, \infty)$
- Minimize: TRUE
- Required prediction: 'response'

Super classes
mlr3::Measure -> mlr3::MeasureClassif -> MeasureClassifCosts

Public fields
normalize (logical(1))
Normalize the costs?
Active bindings

costs (numeric matrix())
    Matrix of costs (truth in columns, predicted response in rows).

Methods

Public methods:

• MeasureClassifCosts$new()
• MeasureClassifCosts$clone()

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

Usage:
MeasureClassifCosts$new()

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

Usage:
MeasureClassifCosts$clone(deep = FALSE)

Arguments:

deepl Whether to make a deep clone.

See Also

Dictionary of Measures: mlr_measures
as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

Other Measure: MeasureClassif,MeasureRegr,Measure,mlr_measures_debug,mlr_measures_elapsed_time,mlr_measures_oob_error,mlr_measures_selected_features,mlr_measures


Examples

# get a cost sensitive task
task = tsk("german_credit")

# cost matrix as given on the UCI page of the german credit data set
# https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)
costs = matrix(c(0, 5, 1, 0), nrow = 2)
dimnames(costs) = list(truth = task$class_names, predicted = task$class_names)
print(costs)

# mlr3 needs truth in columns, predictions in rows
costs = t(costs)

# create measure which calculates the absolute costs
m = msr("classif.costs", id = "german_credit_costs", costs = costs, normalize = FALSE)

# fit models and calculate costs
learner = lrn("classif.rpart")
rr = resample(task, learner, rsmp("cv", folds = 3))
rr$aggregate(m)

mlr_measures_classif.dor

Diagnostic Odds Ratio

Description

Binary classification measure defined as

\[
\frac{TP}{FP} \quad \frac{FN}{TN}
\]

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

mlr_measures$get("dor")
msr("dor")

Meta Information

- Type: "binary"
- Range: [0, \infty)
- Minimize: FALSE
- Required prediction: response

Note

The score function calls mlr3measures::dor() from package mlr3measures.
If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
mlr_measures_classif.fbeta

F-beta Score

Description

Binary classification measure defined with \( P \) as \( \text{precision()} \) and \( R \) as \( \text{recall()} \) as

\[
(1 + \beta^2) \frac{P \cdot R}{(\beta^2 P) + R}.
\]

It measures the effectiveness of retrieval with respect to a user who attaches \( \beta \) times as much importance to recall as precision. For \( \beta = 1 \), this measure is called "F1" score.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function \( \text{msr()} \):

\[
\text{mlr_measures$get("fbeta")}
\]
\[
\text{msr("fbeta")}
\]

Meta Information

- Type: "binary"
- Range: \([0, 1]\)
- Minimize: FALSE
- Required prediction: response
Note
The score function calls `mlr3measures::fbeta()` from package `mlr3measures`.
If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also
Dictionary of Measures: `mlr_measures`
`as.data.table(mlr_measures)` for a complete table of all (also dynamically created) Measure implementations.


---

**mlr_measures_classif.fdr**

*False Discovery Rate*

**Description**

Binary classification measure defined as

\[
\text{FDR} = \frac{\text{FP}}{\text{TP} + \text{FP}}
\]

**Dictionary**

This `Measure` can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("fdr")
msr("fdr")
```
Meta Information

- Type: "binary"
- Range: [0, 1]
- Minimize: TRUE
- Required prediction: response

Note

The score function calls `mlr3measures::fdr()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures` as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


---

**mlr_measures_classif.fn**

*False Negatives*

Description

Classification measure counting the false negatives (type 2 error), i.e. the number of predictions indicating a negative class label while in fact it is positive. This is sometimes also called a "false alarm".
Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("fn")
msr("fn")
```

Meta Information

- Type: "binary"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

Note

The score function calls `mlr3measures::fn()` from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


mlr_measures_classif.fnr

False Negative Rate

Description

Binary classification measure defined as

\[
\frac{FN}{TP + FN}
\]

Also known as "miss rate".

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("fnr")
msr("fnr")
```

Meta Information

- Type: "binary"
- Range: [0, 1]
- Minimize: TRUE
- Required prediction: response

Note

The score function calls mlr3measures::fnr() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

**False Omission Rate**

**Description**

Binary classification measure defined as

\[
\text{FOMR} = \frac{FN}{FN + TN}
\]

**Dictionary**

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function `msr()`:

```r
mlr_measures$get("fomr")
msr("fomr")
```

**Meta Information**

- Type: "binary"
- Range: \([0, 1]\)
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls `mlr3measures::fomr()` from package mlr3measures. If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`. 
See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


---

`mlr_measures_classif.fp`

*False Positives*

**Description**

Classification measure counting the false positives (type 1 error), i.e. the number of predictions indicating a positive class label while in fact it is negative.

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("fp")
msr("fp")
```

**Meta Information**

- **Type:** "binary"
- **Range:** $[0, \infty)$
- **Minimize:** TRUE
- **Required prediction:** response
Note

The score function calls `mlr3measures::fp()` from package `mlr3measures`.

If the measure is undefined for the input, `NaN` is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures`

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


**mlr_measures_classif.fpr**

*False Positive Rate*

**Description**

Binary classification measure defined as

\[
\text{FPR} = \frac{FP}{FP + TN}
\]

Also know as fall out or probability of false alarm.

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("fpr")
msr("fpr")
```
Meta Information

- Type: "binary"
- Range: [0, 1]
- Minimize: TRUE
- Required prediction: response

Note

The score function calls `mlr3measures::fpr()` from package `mlr3measures`.
If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures`
as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


---

\[ -\frac{1}{n} \sum_{i=1}^{n} \log (p_i) \]

where \( p_i \) is the probability for the true class of observation \( i \).
Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("logloss")
msr("logloss")
```

Meta Information

- **Type:** "classif"
- **Range:** \([0, \infty]\)
- **Minimize:** TRUE
- **Required prediction:** prob

Note

The score function calls `mlr3measures::logloss()` from package mlr3measures.
If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


**Description**

Brier score for multi-class classification problems with \( r \) labels defined as

\[
\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{r} (I_{ij} - p_{ij})^2.
\]

\( I_{ij} \) is 1 if observation \( i \) has true label \( j \), and 0 otherwise.

Note that there also is the more common definition of the Brier score for binary classification problems in \texttt{bbrier}().

**Dictionary**

This Measure can be instantiated via the dictionary \texttt{mlr_measures} or with the associated sugar function \texttt{msr()}:

\begin{verbatim}
mlr_measures$get("mbrier")
msr("mbrier")
\end{verbatim}

**Meta Information**

- Type: "classif"
- Range: [0, 2]
- Minimize: TRUE
- Required prediction: prob

**Note**

The score function calls \texttt{mlr3measures::mbrier()} from package \texttt{mlr3measures}.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field \texttt{na_value}.

**See Also**

Dictionary of Measures: \texttt{mlr_measures}

\texttt{as.data.table(mlr_measures)} for a complete table of all (also dynamically created) Measure implementations.


Other multiclass classification measures: \texttt{mlr_measures_classif.acc, mlr_measures_classif.bacc, mlr_measures_classif.ce, mlr_measures_classif.costs, mlr_measures_classif.logloss}
Matthews Correlation Coefficient

Description

Binary classification measure defined as

\[ \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \]

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("mcc")
msr("mcc")
```

Meta Information

- Type: "binary"
- Range: \([-1, 1]\]
- Minimize: FALSE
- Required prediction: response

Note

The score function calls mlr3measures::mcc() from package mlr3measures.
If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.

See Also

Dictionary of Measures: mlr_measures
as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

Other classification measures: mlr_measures_classif.acc, mlr_measures_classif.auc, mlr_measures_classif.bacc,
mlr_measures_classif.bbrier, mlr_measures_classif.ce, mlr_measures_classif.costs,
mlr_measures_classif.dor, mlr_measures_classif.fbeta, mlr_measures_classif.fdr, mlr_measures_classif.fmr,
mlr_measures_classif.fn, mlr_measures_classif.fomr, mlr_measures_classif.fpr, mlr_measures_classif.fp,
mlr_measures_classif.logloss, mlr_measures_classif.mbrier, mlr_measures_classif.npv,
mlr_measures_classif.ppv, mlr_measures_classif.prauc, mlr_measures_classif.precision,
mlr_measures_classif.recall, mlr_measures_classif.sensitivity, mlr_measures_classif.specificity,
mlr_measures_classif.tnr, mlr_measures_classif.tn, mlr_measures_classif.tpr, mlr_measures_classif.tp
mlr_measures_classif.npv

Negative Predictive Value

Description

Binary classification measure defined as

\[
\text{npv} = \frac{\text{TN}}{\text{FN} + \text{TN}}.
\]

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("npv")
msr("npv")
```

Meta Information

- Type: "binary"
- Range: [0, 1]
- Minimize: FALSE
- Required prediction: response

Note

The score function calls `mlr3measures::npv()` from package mlr3measures. If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

Other classification measures: mlr_measures_classif.acc, mlr_measures_classif.auc, mlr_measures_classif.bacc, mlr_measures_classif.bbrier, mlr_measures_classif.ce, mlr_measures_classif.costs,
**mlr_measures_classif.ppv**

Positive Predictive Value

**Description**

Binary classification measure defined as

$$\frac{TP}{TP + FP}.$$  

Also known as "precision".

**Dictionary**

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("ppv")
msr("ppv")
```

**Meta Information**

- **Type**: "binary"
- **Range**: [0, 1]
- **Minimize**: FALSE
- **Required prediction**: response

**Note**

The score function calls `mlr3measures::ppv()` from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


---

mlr_measures_classif.prauc

*Area Under the Precision-Recall Curve*

**Description**

Computes the area under the Precision-Recall curve (PRC). The PRC can be interpreted as the relationship between precision and recall (sensitivity), and is considered to be a more appropriate measure for unbalanced datasets than the ROC curve. The PRC is computed by integration of the piecewise function.

**Dictionary**

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("prauc")
msr("prauc")
```

**Meta Information**

- Type: "binary"
- Range: [0, 1]
- Minimize: FALSE
- Required prediction: prob
Note

The score function calls `mlr3measures::prauc()` from package `mlr3measures`.

If the measure is undefined for the input, `NaN` is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures`

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


---

**mlr_measures_classif.precision**

*Positive Predictive Value*

**Description**

Binary classification measure defined as

\[
\frac{TP}{TP + FP}
\]

Also know as "precision".

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("precision")
msr("precision")
```
**Meta Information**

- Type: "binary"
- Range: [0, 1]
- Minimize: FALSE
- Required prediction: response

**Note**

The score function calls `mlr3measures::precision()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

**See Also**

Dictionary of Measures: `mlr_measures`

`as.data.table(mlr_measures)` for a complete table of all (also dynamically created) Measure implementations.


---

**mlr_measures_classif.recall**

*True Positive Rate*

**Description**

Binary classification measure defined as

\[
\frac{TP}{TP + FN}
\]

Also known as "recall" or "sensitivity".
Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("recall")
msr("recall")
```

Meta Information

- **Type:** "binary"
- **Range:** [0, 1]
- **Minimize:** FALSE
- **Required prediction:** response

Note

The score function calls `mlr3measures::recall()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


Description

Binary classification measure defined as

\[
\text{TP} \over \text{TP + FN}
\]

Also known as "recall" or "sensitivity".

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```
mlr_measures$get("sensitivity")
msr("sensitivity")
```

Meta Information

- Type: "binary"
- Range: \([0, 1]\)
- Minimize: FALSE
- Required prediction: response

Note

The score function calls mlr3measures::sensitivity() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

Other classification measures:


### `mlr_measures_classif.specificity`

**True Negative Rate**

**Description**

Binary classification measure defined as

\[
\frac{\text{TN}}{\text{FP} + \text{TN}}
\]

Also know as "specificity".

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("specificity")
msr("specificity")
```

**Meta Information**

- Type: "binary"
- Range: \([0, 1]\)
- Minimize: FALSE
- Required prediction: response

**Note**

The score function calls `mlr3measures::specificity()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.
See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


Description

Classification measure counting the true negatives, i.e. the number of predictions correctly indicating a negative class label.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

mlr_measures$get("tn")
msr("tn")

Meta Information

- Type: "binary"
- Range: [0, ∞)
- Minimize: FALSE
- Required prediction: response
Note

The score function calls `mlr3measures::tn()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures`
as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


---

**mlr_measures_classif.tnr**

*True Negative Rate*

**Description**

Binary classification measure defined as

\[
\frac{\text{TN}}{\text{FP} + \text{TN}}
\]

Also known as "specificity".

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("tnr")
msr("tnr")
```
Meta Information

- Type: "binary"
- Range: [0, 1]
- Minimize: FALSE
- Required prediction: response

Note

The score function calls `mlr3measures::tnr()` from package `mlr3measures`.
If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures` as.data.table(`mlr_measures`) for a complete table of all (also dynamically created) Measure implementations.


Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("tp")
msr("tp")
```

Meta Information

- Type: "binary"
- Range: \([0, \infty)\)
- Minimize: FALSE
- Required prediction: response

Note

The score function calls `mlr3measures::tp()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures`

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


Description

Binary classification measure defined as

$$ \frac{TP}{TP + FN} $$

Also know as "recall" or "sensitivity".

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("tpr")
msr("tpr")
```

Meta Information

- Type: "binary"
- Range: [0, 1]
- Minimize: FALSE
- Required prediction: response

Note

The score function calls `mlr3measures::tpr()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

Other classification measures:

- mlr_measures_classif.acc
- mlr_measures_classif.auc
- mlr_measures_classif.bacc
- mlr_measures_classif.brier
- mlr_measures_classif.ce
- mlr_measures_classif.costs
- mlr_measures_classif.dor
- mlr_measures_classif.fbeta
- mlr_measures_classif.fdr
- mlr_measures_classif.fnr
- mlr_measures_classif.fn
- mlr_measures_classif.fomr
- mlr_measures_classif.fpr
- mlr_measures_classif.fp
- mlr_measures_classif.logloss
- mlr_measures_classif.mbrier
- mlr_measures_classif.mcc
- mlr_measures_classif.npv
- mlr_measures_classif.ppv
- mlr_measures_classif.prauc
- mlr_measures_classif.precision
- mlr_measures_classif.recall
- mlr_measures_classif.sensitivity
- mlr_measures_classif.specificity
- mlr_measures_classif.tnr
- mlr_measures_classif.tn
- mlr_measures_classif.tp

---

**mlr_measures_debug**

**Debug Measure**

**Description**

This measure returns the number of observations in the Prediction object. Its main purpose is debugging.

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("debug")
msr("debug")
```

**Meta Information**

- Type: NA
- Range: [0, ∞)
- Minimize: NA
- Required prediction: 'response'

**Super class**

`mlr3::Measure` -> `MeasureDebug`

**Public fields**

- `na_ratio (numeric(1))`
  
  Ratio of scores which randomly should be NA, between 0 (default) and 1. Default is 0.

**Methods**

**Public methods:**

- `MeasureDebug$new()`
- `MeasureDebug$clone()`

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

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

Usage:
MeasureDebug$clone(deep = FALSE)

Arguments:
deepe Whether to make a deep clone.

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

Other Measure: MeasureClassif, MeasureRegr, Measure, mlr_measures_classif.costs, mlr_measures_elapsed_time, mlr_measures_oob_error, mlr_measures_selected_features, mlr_measures

Examples

task = tsk("wine")
learner = lrn("classif.featureless")
measure = msr("debug")
rr = resample(task, learner, rsmp("cv", folds = 3))
rr$score(measure)

mlr_measures_elapsed_time

Elapsed Time Measure

Description

Measures the elapsed time during train ("time_train"), predict ("time_predict"), or both ("time_both").

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

mlr_measures$get("time_train")
msr("time_train")

Meta Information

- Type: NA
- Range: [0, ∞)
- Minimize: TRUE
- Required prediction: 'response'
Super class

`mlr3::Measure` -> `MeasureElapsedTime`

Public fields

`s stages (character())`
Which stages of the learner to measure?

Methods

**Public methods:**

- `MeasureElapsedTime$new()`
- `MeasureElapsedTime$clone()`

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

*Usage:*

`MeasureElapsedTime$new(id = "elapsed_time", stages)`

*Arguments:*

- `id (character(1))`
  - Identifier for the new instance.
- `stages (character())`
  - Subset of ("train", "predict"). The runtime of provided stages will be summed.

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

*Usage:*

`MeasureElapsedTime$clone(deep = FALSE)`

*Arguments:*

- `deep` Whether to make a deep clone.

See Also

Dictionary of Measures: `mlr_measures`

`as.data.table(mlr_measures)` for a complete table of all (also dynamically created) Measure implementations.

Other Measure: `MeasureClassif, MeasureRegr, Measure, mlr_measures_classif.costs, mlr_measures_debug, mlr_measures_oob_error, mlr_measures_selected_features, mlr_measures`
Out-of-bag Error Measure

Description

Returns the out-of-bag error of the Learner for learners that support it (learners with property "oob_error"). Returns NA for unsupported learners.

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("oob_error")
msr("oob_error")
```

Meta Information

- Type: NA
- Range: (-\infty, \infty)
- Minimize: NA
- Required prediction: 'response'

Super class

mlr3::Measure -> MeasureOOBError

Methods

Public methods:

- MeasureOOBError$new()
- MeasureOOBError$clone()

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

Usage:

```r
MeasureOOBError$new()
```

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

Usage:

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

Arguments:

depth Whether to make a deep clone.
Bias

Description
Regression measure defined as
\[ \frac{1}{n} \sum_{i=1}^{n} (t_i - r_i). \]

Good predictions score close to 0.

Dictionary
This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("bias")
msr("bias")
```

Meta Information
- Type: "regr"
- Range: \((-\infty, \infty)\)
- Minimize: NA
- Required prediction: response

Note
The score function calls `mlr3measures::bias()` from package `mlr3measures`.
If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also
Dictionary of Measures: mlr_measures
as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.
Other Measure: MeasureClassif, MeasureRegr, Measure, mlr_measures_classif_costs, mlr_measures_debug, mlr_measures_elapsed_time, mlr_measures_selected_features, mlr_measures
mlr_measures_regr.ktau

Kendall’s tau

Description
Regression measure defined as Kendall’s rank correlation coefficient between truth and response. Calls stats::cor() with method set to "kendall".

Dictionary
This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

mlr_measures$get("ktau")
msr("ktau")

Meta Information
- Type: "regr"
- Range: \([-1, 1]\]
- Minimize: FALSE
- Required prediction: response

Note
The score function calls mlr3measures::ktau() from package mlr3measures.
If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
mlr_measures_regr.mae  Mean Absolute Errors

Description

Regression measure defined as

$$\frac{1}{n} \sum_{i=1}^{n} |t_i - r_i|.$$  

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$'get("mae")
msr("mae")
```

Meta Information

- Type: "regr"
- Range: $[0, \infty)$
- Minimize: TRUE
- Required prediction: response

Note

The score function calls mlr3measures::mae() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
mlr_measures_regr.mape

Mean Absolute Percent Error

Description

Regression measure defined as

$$\frac{1}{n} \sum_{i=1}^{n} \frac{|t_i - r_i|}{t_i}$$

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

mlr_measures$get("mape")
msr("mape")

Meta Information

- Type: "regr"
- Range: $[0, \infty)$
- Minimize: TRUE
- Required prediction: response

Note

The score function calls mlr3measures::mape() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
**mlr_measures_regr.maxae**

**Max Absolute Error**

**Description**

Regression measure defined as

\[ \max |t_i - r_i|. \]

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("maxae")
msr("maxae")
```

**Meta Information**

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls `mlr3measures::maxae()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.

See Also

Dictionary of Measures: `mlr_measures`
as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

mlr_measures_regr.medae

Median Absolute Errors

Description

Regression measure defined as
\[
\text{median} \sum_{i} |t_i - r_i |
\]

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("medae")
msr("medae")
```

Meta Information

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

Note

The score function calls `mlr3measures::medae()` from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`.
See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


mlr_measures_regr.medse

Median Squared Error

Description

Regression measure defined as

\[
\text{Median} \left( (t_i - r_i)^2 \right).
\]

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("medse")
msr("medse")
```

Meta Information

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

Note

The score function calls mlr3measures::medse() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


---

**mlr_measures_regr.mse**  Mean Squared Error

**Description**

Regression measure defined as

\[
\frac{1}{n} \sum_{i=1}^{n} (t_i - r_i)^2.
\]

**Dictionary**

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$pdf("mse")
msr("mse")
```

**Meta Information**

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls mlr3measures::mse() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
mlr_measures_regr.msle

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


mlr_measures_regr.msle

*Mean Squared Log Error*

**Description**

Regression measure defined as

\[
\frac{1}{n} \sum_{i=1}^{n} \left( \ln(1 + t_i) - \ln(1 + r_i) \right)^2.
\]

**Dictionary**

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

mlr_measures$get("msle")
msr("msle")

**Meta Information**

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls mlr3measures::msle() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


---

**Percent Bias**

**Description**

Regression measure defined as

\[ \frac{1}{n} \sum_{i=1}^{n} \frac{(t_i - r_i)}{|t_i|}. \]

Good predictions score close to 0.

**Dictionary**

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

mlr_measures$get("pbias")
msr("pbias")

**Meta Information**

- Type: "regr"
- Range: \((-\infty, \infty)\)
- Minimize: NA
- Required prediction: response

**Note**

The score function calls mlr3measures::pbias() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
**mlr_measures_regr.rae  Relative Absolute Error**

**Description**

Regression measure defined as

\[
\frac{\sum_{i=1}^{n} |t_i - r_i|}{\sum_{i=1}^{n} |t_i - \bar{t}|}.
\]

Can be interpreted as absolute error of the predictions relative to a naive model predicting the mean.

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("rae")
msr("rae")
```

**Meta Information**

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls `mlr3measures::rae()` from package `mlr3measures`.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`. 

---

**See Also**

Dictionary of Measures: `mlr_measures`

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


---

**mlr_measures_regr.rmse**

*Root Mean Squared Error*

**Description**

Regression measure defined as

\[
\sqrt{\frac{1}{n} \sum_{i=1}^{n} (t_i - r_i)^2}.
\]

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("rmse")
msr("rmse")
```

**Meta Information**

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls `mlr3measures::rmse()` from package `mlr3measures`.

If the measure is undefined for the input, `NaN` is returned. This can be customized by setting the field `na_value`. 
**See Also**

Dictionary of Measures: `mlr_measures`

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


---

**mlr_measures_regr.rmsle**

*Root Mean Squared Log Error*

**Description**

Regression measure defined as

\[
\sqrt{\frac{1}{n} \sum_{i=1}^{n} (\ln(1 + t_i) - \ln(1 + r_i))^2}.
\]

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("rmsle")
msr("rmsle")
```

**Meta Information**

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: `TRUE`
- Required prediction: `response`

**Note**

The score function calls `mlr3measures::rmsle()` from package `mlr3measures`.

If the measure is undefined for the input, `NaN` is returned. This can be customized by setting the field `na_value`.
See Also

Dictionary of Measures: mlr_measures
as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


mlr_measures_regr.rrse

*Root Relative Squared Error*

**Description**

Regression measure defined as

$$\sqrt{\frac{\sum_{i=1}^{n} (t_i - r_i)^2}{\sum_{i=1}^{n} (t_i - \bar{t})^2}}.$$

Can be interpreted as root of the squared error of the predictions relative to a naive model predicting the mean.

**Dictionary**

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("rrse")
msr("rrse")
```

**Meta Information**

- Type: "regr"
- Range: [0, \(\infty\))
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls mlr3measures::rrse() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
**mlr_measures_regr.rse**  

**Relative Squared Error**

**Description**

Regression measure defined as

\[
\frac{\sum_{i=1}^{n} (t_i - r_i)^2}{\sum_{i=1}^{n} (t_i - \bar{t})^2}.
\]

Can be interpreted as squared error of the predictions relative to a naive model predicting the mean.

**Dictionary**

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("rse")
msr("rse")
```

**Meta Information**

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls mlr3measures::rse() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
mlr_measures_regr.rsq  

R Squared

Description
Regression measure defined as

\[ 1 - \frac{\sum_{i=1}^{n} (t_i - r_i)^2}{\sum_{i=1}^{n} (t_i - \bar{t})^2}. \]

Also known as coefficient of determination or explained variation. Subtracts the rse() from 1, hence it compares the squared error of the predictions relative to a naive model predicting the mean.

Dictionary
This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function m$sr()$:

```r
mlr_measures$get("rsq")
msr("rsq")
```

Meta Information
- Type: "regr"
- Range: (−∞, 1]
- Minimize: FALSE
- Required prediction: response

Note
The score function calls mlr3measures::rsq() from package mlr3measures.
If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
**mlr_measures_regr.sae**  
*Sum of Absolute Errors*

**Description**

Regression measure defined as

\[
\sum_{i=1}^{n} |t_i - r_i|.
\]

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("sae")
msr("sae")
```

**Meta Information**

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls `mlr3measures::sae()` from package `mlr3measures`. If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`. 

**See Also**

Dictionary of Measures: `mlr_measures`  as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

**Symmetric Mean Absolute Percent Error**

**Description**

Regression measure defined as

\[ \frac{2}{n} \sum_{i=1}^{n} \frac{|t_i - r_i|}{|t_i| + |r_i|} \]

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```
mlr_measures$get("smape")
msr("smape")
```

**Meta Information**

- Type: "regr"
- Range: \([0, 2]\)
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls `mlr3::smape()` from package `mlr3`. If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`. 

See Also

Dictionary of Measures: `mlr_measures`
See Also

Dictionary of Measures: mlr_measures

See Also

Dictionary of Measures: mlr_measures

as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.


mlr_measures_regr.srho

Spearman’s rho

Description

Regression measures defined as Spearman’s rank correlation coefficient between truth and response. Calls stats::cor() with method set to "spearman".

Dictionary

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

mlr_measures$get("srho")
msr("srho")

Meta Information

- Type: "regr"
- Range: \([-1, 1]\]
- Minimize: FALSE
- Required prediction: response

Note

The score function calls mlr3measures::srho() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
**mlr_measures_regr.sse**

Sum of Squared Errors

**Description**

Regression measure defined as

\[
\sum_{i=1}^{n} (t_i - r_i)^2.
\]

**Dictionary**

This Measure can be instantiated via the dictionary `mlr_measures` or with the associated sugar function `msr()`:

```r
mlr_measures$get("sse")
msr("sse")
```

**Meta Information**

- Type: "regr"
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: response

**Note**

The score function calls `mlr3measures::sse()` from package `mlr3measures`. If the measure is undefined for the input, NaN is returned. This can be customized by setting the field `na_value`. 

See Also

Dictionary of Measures: `mlr_measures` as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.

**Selected Features Measure**

**Description**

Measures the number of selected features by extracting it from learners with property "selected_features". If the learner does not support this, NA is returned.

This measure requires the Task and the Learner for scoring.

**Dictionary**

This Measure can be instantiated via the dictionary mlr_measures or with the associated sugar function msr():

```r
mlr_measures$get("selected_features")
msr("selected_features")
```

**Meta Information**

- **Type:** NA
- **Range:** [0, ∞)
- **Minimize:** TRUE
- **Required prediction:** 'response'

**Super class**

mlr3::Measure -> MeasureSelectedFeatures

**Public fields**

- `normalize` (logical(1))
  
  If set to TRUE, divides the number of features by the total number of features.
Methods

Public methods:

• MeasureSelectedFeatures$new()
• MeasureSelectedFeatures$clone()

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

Usage:
MeasureSelectedFeatures$new(normalize = FALSE)

Arguments:
normalize (logical(1))
If set to TRUE, divides the number of features by the total number of features.

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

Usage:
MeasureSelectedFeatures$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

See Also

Dictionary of Measures: mlr_measures
as.data.table(mlr_measures) for a complete table of all (also dynamically created) Measure implementations.
Other Measure: MeasureClassif, MeasureRegr, Measure, mlr_measures_classif.costs, mlr_measures_debug, mlr_measures_elapsed_time, mlr_measures_oob_error, mlr_measures

Description

A simple mlr3misc::Dictionary storing objects of class Resampling. Each resampling has an associated help page, see mlr_resamplings_[id].
This dictionary can get populated with additional resampling strategies by add-on packages.
For a more convenient way to retrieve and construct resampling strategies, see rsmp() / rsmps().

Format

R6::R6Class object inheriting from mlr3misc::Dictionary.

Methods

See mlr3misc::Dictionary.
**MLR Resamplings Bootstrap**

### Description

Splits data into bootstrap samples (sampling with replacement). Hyperparameters are the number of bootstrap iterations (`repeats`, default: 30) and the ratio of observations to draw per iteration (`ratio`, default: 1) for the training set.

### Dictionary

This Resampling can be instantiated via the dictionary `mlr_resamplings` or with the associated sugar function `rsmp()`:

```r
mlr_resamplings$get("bootstrap")
rsmp("bootstrap")
```

### Parameters

- **repeats**: `integer(1)`
  - Number of repetitions.

- **ratio**: `numeric(1)`
  - Ratio of observations to put into the training set.

### Super class

- `mlr3::Resampling` - `ResamplingBootstrap`
Active bindings

`iters (integer(1))`
Returns the number of resampling iterations, depending on the values stored in the `param_set`.

Methods

**Public methods:**

- `ResamplingBootstrap$new()`
- `ResamplingBootstrap$clone()`

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

*Usage:*

```r
ResamplingBootstrap$new()
```

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

*Usage:*

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

*Arguments:*

- `deep` Whether to make a deep clone.

References


See Also

- Dictionary of Resamplings: `mlr_resamplings`

  `as.data.table(mlr_resamplings)` for a complete table of all (also dynamically created) Resampling implementations.

  Other Resampling: `Resampling, mlr_resamplings_custom, mlr_resamplings_cv, mlr_resamplings_holdout, mlr_resamplings_insample, mlr_resamplings_loo, mlr_resamplings_repeated_cv, mlr_resamplings_subsampling, mlr_resamplings`

Examples

```r
# Create a task with 10 observations
task = tsk("penguins")
task$filter(1:10)

# Instantiate Resampling
rb = rsmp("bootstrap", repeats = 2, ratio = 1)
rb$instantiate(task)

# Individual sets:
brb$train_set(1)
```
Custom Resampling

Description
Splits data into training and test sets using manually provided indices.

Dictionary
This Resampling can be instantiated via the dictionary mlr_resamplings or with the associated sugar function rsmp():

```r
mlr_resamplings$get("custom")
rsp("custom")
```

Super class
mlr3::Resampling -> ResamplingCustom

Active bindings
- `iters` (integer(1))
  Returns the number of resampling iterations, depending on the values stored in the param_set.
- `hash` (character(1))
  Hash (unique identifier) for this object.

Methods
- **Public methods:**
  - `ResamplingCustom$new()`
  - `ResamplingCustom$instantiate()`
  - `ResamplingCustom$clone()`

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

Usage:
```r
ResamplingCustom$new()
```

**Method** `instantiate()`: Instantiate this Resampling with custom splits into training and test set.

Usage:
ResamplingCustom$instantiate(task, train_sets, test_sets)

**Arguments:**

- **task**  
  Task  
  Mainly used to check if `train_sets` and `test_sets` are feasible.
- **train_sets**  
  (list of integer())  
  List with row ids for training, one list element per iteration. Must have the same length as `test_sets`.
- **test_sets**  
  (list of integer())  
  List with row ids for testing, one list element per iteration. Must have the same length as `train_sets`.

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

**Usage:**

ResamplingCustom$clone(deep = FALSE)

**Arguments:**

- **deep**  
  Whether to make a deep clone.

**See Also**

- Dictionary of Resamplings: `mlr_resamplings`
- `as.data.table(mlr_resamplings)` for a complete table of all (also dynamically created) Resampling implementations.
- Other Resampling: `Resampling, mlr_resamplings_bootstrap, mlr_resamplings_cv, mlr_resamplings_holdout, mlr_resamplings_insample, mlr_resamplings_loo, mlr_resamplings_repeated_cv, mlr_resamplings_subsampling, mlr_resamplings`

**Examples**

# Create a task with 10 observations
```r
task = tsk("penguins")
task$filter(1:10)
```

# Instantiate Resampling
```r
rc = rsmp("custom")
train_sets = list(1:5, 5:10)
test_sets = list(5:10, 1:5)
rc$instantiate(task, train_sets, test_sets)
```

```r
rc$train_set(1)
rc$test_set(1)
```
**mlr_resamplings_cv**  
*Cross-Validation Resampling*

**Description**
Splits data using a folds-folds (default: 10 folds) cross-validation.

**Dictionary**
This Resampling can be instantiated via the dictionary `mlr_resamplings` or with the associated sugar function `rsmp()`:

```
mlr_resamplings$get("cv")
rsmp("cv")
```

**Parameters**
- `folds` (integer(1))  
  Number of folds.

**Super class**
`mlr3::Resampling` -> `ResamplingCV`

**Active bindings**
- `iters` (integer(1))  
  Returns the number of resampling iterations, depending on the values stored in the `param_set`.

**Methods**
- **Public methods:**
  - `ResamplingCV$new()`
  - `ResamplingCV$clone()`

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

**Method** `clone()`: The objects of this class are cloneable with this method.  
*Usage:*
```
ResamplingCV$clone(deep = FALSE)
```
*Arguments:*
- `deep`  
  Whether to make a deep clone.
References


See Also

Dictionary of Resamplings: mlr_resamplings

as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) Resampling implementations.

Other Resampling: Resampling, mlr_resamplings_bootstrap, mlr_resamplings_custom, mlr_resamplings_holdout, mlr_resamplings_insample, mlr_resamplings_loo, mlr_resamplings_repeated_cv, mlr_resamplings_subsampling, mlr_resamplings

Examples

```r
# Create a task with 10 observations
task = tsk("penguins")
task$filter(1:10)

# Instantiate Resampling
rcv = rsmp("cv", folds = 3)
rcv$instantiate(task)

# Individual sets:
rcv$train_set(1)
rcv$test_set(1)
intersect(rcv$train_set(1), rcv$test_set(1))

# Internal storage:
rcv$instance # table
```

---

**mlr_resamplings_holdout**

*Holdout Resampling*

Description

Splits data into a training set and a test set. Parameter *ratio* determines the ratio of observation going into the training set (default: 2/3).

Dictionary

This Resampling can be instantiated via the dictionary mlr_resamplings or with the associated sugar function rsmp():

```r
mlr_resamplings$get("holdout")
rsmp("holdout")
```
Parameters

- **ratio (numeric(1))**
  Ratio of observations to put into the training set.

Super class

`mlr3::Resampling` -> `ResamplingHoldout`

Public fields

- **iters (integer(1))**
  Returns the number of resampling iterations, depending on the values stored in the `param_set`.

Methods

Public methods:

- `ResamplingHoldout$new()`
- `ResamplingHoldout$clone()`

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

Usage:

```
ResamplingHoldout$new()
```

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

Usage:

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

Arguments:

depth  Whether to make a deep clone.

References


See Also

Dictionary of Resamplings: `mlr_resamplings`
as.data.table(`mlr_resamplings`) for a complete table of all (also dynamically created) Resampling implementations.

Other Resampling: `Resampling`, `mlr_resamplings_bootstrap`, `mlr_resamplings_custom`, `mlr_resamplings_cv`, `mlr_resamplings_insample`, `mlr_resamplings_loo`, `mlr_resamplings_repeated_cv`, `mlr_resamplings_subsampling`, `mlr_resamplings`
Examples

# Create a task with 10 observations
task = tsk("penguins")
task$filter(1:10)

# Instantiate Resampling
rho = rsmp("holdout", ratio = 0.5)
rho$instantiate(task)

# Individual sets:
rho$train_set(1)
rho$test_set(1)
intersect(rho$train_set(1), rho$test_set(1))

# Internal storage:
rho$instance # simple list

---

**mlr_resamplings_insample**

*Insample Resampling*

---

**Description**

Uses all observations as training and as test set.

**Dictionary**

This **Resampling** can be instantiated via the dictionary **mlr_resamplings** or with the associated sugar function **rsmp()**:

```r
mlr_resamplings$get("insample")
rspm("insample")
```

**Super class**

```r
mlr3::Resampling -> ResamplingInsample
```

**Public fields**

`iters (integer(1))`

Returns the number of resampling iterations, depending on the values stored in the `param_set`.

**Methods**

Public methods:

- `ResamplingInsample$new()`
- `ResamplingInsample$clone()`
**Method** `new()`: Creates a new instance of this R6 class.

*Usage:*

```r
ResamplingInsample$new()
```

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

*Usage:*

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

*Arguments:*

- `deep` Whether to make a deep clone.

---

**See Also**

- Dictionary of Resamplings: `mlr_resamplings`
- `as.data.table(mlr_resamplings)` for a complete table of all (also dynamically created) Resampling implementations.

Other Resampling: `Resampling, mlr_resamplings_bootstrap, mlr_resamplings_custom, mlr_resamplings_cv, mlr_resamplings_holdout, mlr_resamplings_loo, mlr_resamplings_repeated_cv, mlr_resamplings_subsampling, mlr_resamplings`

---

**Examples**

```r
# Create a task with 10 observations
task = tsk("penguins")
task$filter(1:10)

# Instantiate Resampling
rins = rsmp("insample")
rins$instantiate(task)

rins$train_set(1)
rins$test_set(1)

# Internal storage:
rins$instance # just row ids
```

---

**mlr_resamplings_loo**  
*Leave-One-Out Cross-Validation*

---

**Description**

Splits data using leave-one-observation-out. This is identical to cross-validation with the number of folds set to the number of observations.
Dictionary

This Resampling can be instantiated via the dictionary mlr_resamplings or with the associated sugar function rsmp():

```r
mlr_resamplings$get("loo")
rsp("loo")
```

Super class

mlr3::Resampling -> ResamplingLOO

Active bindings

`iters` (integer(1))

Returns the number of resampling iterations which is the number of rows of the task provided to instantiate. Is NA if the resampling has not been instantiated.

Methods

Public methods:

- `ResamplingLOO$new()`
- `ResamplingLOO$clone()`

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

Usage:

```r
ResamplingLOO$new()
```

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

Usage:

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

Arguments:

- `deep` Whether to make a deep clone.

References


See Also

Dictionary of Resamplings: `mlr_resamplings`

as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) Resampling implementations.

Other Resampling: `Resampling,mlr_resamplings_bootstrap,mlr_resamplings_custom,mlr_resamplings_cv,mlr_resamplings_holdout,mlr_resamplings_insample,mlr_resamplings_repeated_cv,mlr_resamplings_subsampling,mlr_resamplings`
Examples

# Create a task with 10 observations
task = tsk("penguins")
task$filter(1:10)

# Instantiate Resampling
rcv = rsmp("loo")
rcv$instantiate(task)

# Individual sets:
rcv$train_set(1)
rcv$test_set(1)
intersect(rcv$train_set(1), rcv$test_set(1))

# Internal storage:
rcv$instance # vector

mlr_resamplings_repeated_cv

Repeated Cross-Validation Resampling

Description

Splits data repeats (default: 10) times using a folds-fold (default: 10) cross-validation.

The iteration counter translates to repeats blocks of folds cross-validations, i.e., the first folds iterations belong to a single cross-validation.

Iteration numbers can be translated into folds or repeats with provided methods.

Dictionary

This Resampling can be instantiated via the dictionary mlr_resamplings or with the associated sugar function rsmp():

mlr_resamplings$get("repeated_cv")
rmsp("repeated_cv")

Parameters

- repeats (integer(1))
  Number of repetitions.
- folds (integer(1))
  Number of folds.

Super class

mlr3::Resampling -> ResamplingRepeatedCV
Active bindings

`iters` (integer(1))

Returns the number of resampling iterations, depending on the values stored in the `param_set`.

Methods

**Public methods:**
- `ResamplingRepeatedCV$new()`
- `ResamplingRepeatedCV$folds()`
- `ResamplingRepeatedCV$repeats()`
- `ResamplingRepeatedCV$clone()`

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

**Usage:**

```r
ResamplingRepeatedCV$new()
```

**Method `folds()`**: Translates iteration numbers to fold numbers.

**Usage:**

```r
ResamplingRepeatedCV$folds(iters)
```

**Arguments:**

- `iters` (integer())
  - Iteration number.

**Returns**: integer() of fold numbers.

**Method `repeats()`**: Translates iteration numbers to repetition numbers.

**Usage:**

```r
ResamplingRepeatedCV$repeats(iters)
```

**Arguments:**

- `iters` (integer())
  - Iteration number.

**Returns**: integer() of repetition numbers.

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

**Usage:**

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

**Arguments:**

- `deep` Whether to make a deep clone.

References

mlr_resamplings_subsampling

See Also

Dictionary of Resamplings: mlr_resamplings

as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) Resampling implementations.

Other Resampling: Resampling, mlr_resamplings_bootstrap, mlr_resamplings_custom, mlr_resamplings_cv, mlr_resamplings_holdout, mlr_resamplings_insample, mlr_resamplings_loo, mlr_resamplings_subsampling, mlr_resamplings

Examples

# Create a task with 10 observations
task = tsk("penguins")
task$filter(1:10)

# Instantiate Resampling
rrcv = rsmp("repeated_cv", repeats = 2, folds = 3)
rrcv$instantiate(task)
rrcv$iters
rrcv$folds(1:6)
rrcv$repeats(1:6)

# Individual sets:
rrcv$train_set(1)
rrcv$test_set(1)
intersect(rrcv$train_set(1), rrcv$test_set(1))

# Internal storage:
rrcv$instance # table

mlr_resamplings_subsampling

Subsampling Resampling

Description

Splits data repeats (default: 30) times into training and test set with a ratio of ratio (default: 2/3) observations going into the training set.

Dictionary

This Resampling can be instantiated via the dictionary mlr_resamplings or with the associated sugar function rsmp():

mlr_resamplings$get("holdout")
rsmp("holdout")
Parameters

- repeats (integer(1))
  Number of repetitions.
- ratio (numeric(1))
  Ratio of observations to put into the training set.

Super class

`mlr3::Resampling` -> ResamplingSubsampling

Active bindings

iters (integer(1))
Returns the number of resampling iterations, depending on the values stored in the param_set.

Methods

Public methods:

- ResamplingSubsampling$new()
- ResamplingSubsampling$clone()

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

Usage:
ResamplingSubsampling$new()

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

Usage:
ResamplingSubsampling$clone(deep = FALSE)

Arguments:

depth Whether to make a deep clone.

References


See Also

Dictionary of Resamplings: `mlr_resamplings`

as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) Resampling implementations.

Other Resampling: Resampling, mlr_resamplings_bootstrap, mlr_resamplings_custom, mlr_resamplings_cv, mlr_resamplings_holdout, mlr_resamplings_insample, mlr_resamplings_loo, mlr_resamplings_repeated_cv, mlr_resamplings_subsampling
Examples

```r
# Create a task with 10 observations
task = tsk("penguins")
task$filter(1:10)

# Instantiate Resampling
rss = rsm("subsampling", repeats = 2, ratio = 0.5)
rss$instance(task)

# Individual sets:
rss$train_set(1)
rss$test_set(1)
intersect(rss$train_set(1), rss$test_set(1))

# Internal storage:
rss$instance$train # list of index vectors
```

---

**mlr_sugar**

*Syntactic Sugar for Object Construction*

**Description**

Functions to retrieve objects, set hyperparameters and assign to fields in one go. Relies on `mlr3misc::dictionary_sugar_get()` to extract objects from the respective `mlr3misc::Dictionary`:

- `tsk()` for a `Task` from `mlr_tasks`.
- `tsks()` for a list of `Tasks` from `mlr_tasks`.
- `tgen()` for a `TaskGenerator` from `mlr_task_generators`.
- `tgens()` for a list of `TaskGenerators` from `mlr_task_generators`.
- `lrn()` for a `Learner` from `mlr_learners`.
- `lrns()` for a list of `Learners` from `mlr_learners`.
- `rsm()` for a `Resampling` from `mlr_resamplings`.
- `rsmps()` for a list of `Resamplings` from `mlr_resamplings`.
- `msr()` for a `Measure` from `mlr_measures`.
- `msrs()` for a list of `Measures` from `mlr_measures`.

**Usage**

```r
tsk(.key, ...)

tsk(.keys, ...)

tgen(.key, ...)

tgens(.keys, ...)
```
mlr_tasks

lrn(.key, ...)
lrns(.keys, ...)
rsmp(.key, ...)
rsmps(.keys, ...)
msr(.key, ...)
msrs(.keys, ...)

Arguments

- `.key` (character(1))
  Key passed to the respective dictionary to retrieve the object.
- `...` (named list())
  Named arguments passed to the constructor, to be set as parameters in the paradox::ParamSet, or to be set as public field. See mlr3misc::dictionary_sugar_get() for more details.
- `.keys` (character())
  Keys passed to the respective dictionary to retrieve multiple objects.

Value

R6::R6Class object of the respective type, or a list of R6::R6Class objects for the plural versions.

Examples

```r
# penguins task with new id
tsk("penguins", id = "penguins2")

# classification tree with different hyperparameters
# and predict type set to predict probabilities
lrn("classif.rpart", cp = 0.1, predict_type = "prob")

# multiple learners with predict type 'prob'
lrns(c("classif.featureless", "classif.rpart"), predict_type = "prob")
```
mlr_tasks

Description

A simple mlr3misc::Dictionary storing objects of class Task. Each task has an associated help page, see mlr_tasks_[id].

This dictionary can get populated with additional tasks by add-on packages, e.g. mlr3data, mlr3proba or mlr3cluster. mlr3oml allows to interact with OpenML.

For a more convenient way to retrieve and construct tasks, see tsk()/tsks().

Format

R6::R6Class object inheriting from mlr3misc::Dictionary.

Methods

See mlr3misc::Dictionary.

S3 methods

• as.data.table(dict)
  
  Returns a data.table::data.table() with columns "key", "task_type", "measures", "nrow", "ncol" and the number of features of type "lgl", "int", "dbl", "chr", "fct" and "ord" as columns.

See Also

Sugar functions: tsk(), tsks()

Extension Packages: mlr3data

Other Dictionary: mlr_learners, mlr_measures, mlr_resamplings, mlr_task_generators

Other Task: TaskClassif, TaskRegr, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing, mlr_tasks_breast_cancer, mlr_tasks_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_penguins, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo

Examples

as.data.table(mlr_tasks)
task = mlr_tasks$get("penguins") # same as tsk("penguins")
head(task$data())

# Add a new task, based on a subset of penguins:
data = palmerpenguins::penguins
data$species = factor(ifelse(data$species == "Adelie", "1", "0"))
task = TaskClassif$new("penguins.binary", data, target = "species", positive = "1")

# add to dictionary
mlr_tasks$add("penguins.binary", task)

# list available tasks
mlr_tasks$keys()
# retrieve from dictionary
mlr_tasks$get("penguins.binary")

# remove task again
mlr_tasks$remove("penguins.binary")

---

**mlr_tasks_boston_housing**

**Boston Housing Regression Task**

### Description

A regression task for the `mlbench::BostonHousing2` data set.

### Format

`R6::R6Class` inheriting from `TaskRegr`.

### Construction

```r
mlr_tasks$get("boston_housing")
tsk("boston_housing")
```

### Meta Information

- Task type: "regr"
- Dimensions: 506x19
- Properties: -
- Has Missings: `FALSE`
- Target: "medv"

### See Also

- Dictionary of Tasks: `mlr_tasks`
- `as.data.table(mlr_tasks)` for a complete table of all (also dynamically created) Tasks.
Wisconsin Breast Cancer Classification Task

Description

A classification task for the mlbench::BreastCancer data set.

- Column "Id" has been removed.
- Column names have been converted to snake_case.
- Positive class is set to "malignant".
- 16 incomplete cases have been removed from the data set.
- All factor features have been converted to ordered factors.

Format

R6::R6Class inheriting from TaskClassif.

Construction

mlr_tasks$get("breast_cancer")
tsk("breast_cancer")

Meta Information

- Task type: "classif"
- Dimensions: 683x10
- Properties: "twoclass"
- Has Missings: FALSE
- Target: "class"
- Features: "bare_nuclei", "bl_cromatin", "cell_shape", "cell_size", "cl_thickness", "epith_e_size", "marg_adhesion", "mitoses", "normal_nucleoli"

See Also

Dictionary of Tasks: mlr_tasks
as.data.table(mlr_tasks) for a complete table of all (also dynamically created) Tasks.

Other Task: TaskClassif, TaskRegr, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing, mlr_tasks_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_penguins, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo, mlr_tasks
**German Credit Classification Task**

**Description**

A classification task for the German credit data set. The aim is to predict creditworthiness, labeled as "good" and "bad". Positive class is set to label "good".

See example for the creation of a `MeasureClassifCosts` as described misclassification costs.

**Format**

`R6::R6Class inheriting from TaskClassif`.

**Construction**

```r
mlr_tasks$get("german_credit")
tsk("german_credit")
```

**Meta Information**

- Task type: "classif"
- Dimensions: 1000x21
- Properties: "twoclass"
- Has Missings: FALSE
- Target: "credit_risk"
- Features: "age", "amount", "credit_history", "duration", "employment_duration", "foreign_worker", "housing", "installment_rate", "job", "number_credits", "other_debtors", "other_installment_plans", "peopleliable", "personal_status_sex", "present_residence", "property", "purpose", "savings", "status", "telephone"

**Source**

Data set originally published on UCI. This is the preprocessed version taken from package `rchallenge` with factors instead of dummy variables, and corrected as proposed by Ulrike Grömping.

Donor: Professor Dr. Hans Hofmann
Institut für Statistik und Ökonometrie
Universität Hamburg
FB Wirtschaftswissenschaften
Von-Melle-Park 5
2000 Hamburg 13

**References**

mlr_tasks_iris

Iris Classification Task

Description

A classification task for the popular datasets::iris data set.

Format

R6::R6Class inheriting from TaskClassif.

Construction

mlr_tasks$get("iris")
tsk("iris")

Meta Information

- Task type: “classif”
- Dimensions: 150x5
- Properties: “multiclass”
- Has Missings: FALSE
- Target: “Species”

Source


mlr_tasks_mtcars

Motor Trend Regression Task

Description

A regression task for the datasets::mtcars data set. Target variable is mpg (Miles/(US) gallon). Rownames are stored as variable `..rownames` with column role "model".

Format

R6::R6Class inheriting from TaskRegr.

Construction

```r
mlr_tasks$get("mtcars")
tsk("mtcars")
```

Meta Information

- Task type: "regr"
- Dimensions: 32x11
- Properties: -
- Has Missings: FALSE
- Target: "mpg"
- Features: "am", "carb", "cyl", "disp", "drat", "gear", "hp", "qsec", "vs", "wt"

See Also

Dictionary of Tasks: mlr_tasks
as.data.table(mlr_tasks) for a complete table of all (also dynamically created) Tasks.
Other Task: TaskClassif, TaskRegr, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing, mlr_tasks_breast_cancer, mlr_tasks_german_credit, mlr_tasks_mtcars, mlr_tasks_penguins, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo, mlr_tasks
**mlr_tasks_penguins**  
*Palmer Penguins Data Set*

**Description**

Classification data to predict the species of penguins from the `palmerpenguins` package, see `palmerpenguins::penguins`. A better alternative to the *iris* data set.

**Format**

R6::R6Class inheriting from TaskClassif.

**Construction**

```r
mlr_tasks$get("penguins")  
tsk("penguins")
```

**Meta Information**

- Task type: “classif”
- Dimensions: 344x8
- Properties: “multiclass”
- Has Missings: TRUE
- Target: “species”

**Pre-processing**

- The unit of measurement have been removed from the column names. Lengths are given in millimeters (mm), weight in gram (g).

**Source**

`palmerpenguins`

**References**


[https://github.com/allisonhorst/palmerpenguins](https://github.com/allisonhorst/palmerpenguins)
mlr_tasks_pima

See Also

Dictionary of Tasks: mlr_tasks

as.data.table(mlr_tasks) for a complete table of all (also dynamically created) Tasks.

Other Task: TaskClassif, TaskRegr, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing, mlr_tasks_breast_cancer, mlr_tasks_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo, mlr_tasks

---

mlr_tasks_pima  Pima Indian Diabetes Classification Task

Description

A classification task for the mlbench::PimaIndiansDiabetes2 data set. Positive class is set to "pos".

Format

R6::R6Class inheriting from TaskClassif.

Construction

mlr_tasks$get("pima")
tsk("pima")

Meta Information

- Task type: "classif"
- Dimensions: 768x9
- Properties: "twoclass"
- Has Missings: TRUE
- Target: "diabetes"
- Features: "age", "glucose", "insulin", "mass", "pedigree", "pregnant", "pressure", "triceps"

See Also

Dictionary of Tasks: mlr_tasks

as.data.table(mlr_tasks) for a complete table of all (also dynamically created) Tasks.

Other Task: TaskClassif, TaskRegr, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing, mlr_tasks_breast_cancer, mlr_tasks_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_penguins, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo, mlr_tasks
Sonar Classification Task

Description

A classification task for the mlbench::Sonar data set. Positive class is set to "M" (Mine).

Format

R6::R6Class inheriting from TaskClassif.

Construction

mlr_tasks$get("sonar")
tsk("sonar")

Meta Information

- Task type: "classif"
- Dimensions: 208x61
- Properties: "twoclass"
- Has Missings: FALSE
- Target: "Class"

See Also

Dictionary of Tasks: mlr_tasks

as.data.table(mlr_tasks) for a complete table of all (also dynamically created) Tasks.

Other Task: TaskClassif, TaskRegr, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing, mlr_tasks_breast_cancer, mlr_tasks_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_penguins, mlr_tasks_pima, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo, mlr_tasks
Spam Classification Task

Description

Spam data set from the UCI machine learning repository (http://archive.ics.uci.edu/ml/datasets/spambase). Data set collected at Hewlett-Packard Labs to classify emails as spam or non-spam. 57 variables indicate the frequency of certain words and characters in the e-mail. The positive class is set to “spam”.

Format

R6::R6Class inheriting from TaskClassif.

Construction

mlr_tasks$get("spam")
tsks(“spam”)

Meta Information

- Task type: “classif”
- Dimensions: 4601x58
- Properties: “twoclass”
- Has Missings: FALSE
- Target: “type”

Source


Donor: George Forman (gforman@nospam.hpl.hp.com) 650-857-7835

Preprocessing: Columns have been renamed. Preprocessed data taken from the kernlab package.

References

**See Also**

Dictionary of Tasks: `mlr_tasks`  
as.data.table(mlr_tasks) for a complete table of all (also dynamically created) Tasks.

Other Task: `TaskClassif`, `TaskRegr`, `TaskSupervised`, `TaskUnsupervised`, `Task`, `mlr_tasks_boston_housing`,  
`mlr_tasks_breast_cancer`, `mlr_tasks_german_credit`, `mlr_tasks_iris`, `mlr_tasks_mtcars`,  
`mlr_tasks_penguins`, `mlr_tasks_pima`, `mlr_tasks_sonar`, `mlr_tasks_wine`, `mlr_tasks_zoo`,  
`mlr_tasks`

---

**mlr_tasks_wine**  
Wine Classification Task

**Description**

Wine data set from the UCI machine learning repository (https://archive.ics.uci.edu/ml/datasets/wine). Results of a chemical analysis of three types of wines grown in the same region in Italy but derived from three different cultivars.

**Format**

R6::R6Class inheriting from TaskClassif.

**Construction**

mlr_tasks$get("wine")  
tsk("wine")

**Meta Information**

- Task type: “classif”
- Dimensions: 178x14
- Properties: “multiclass”
- Has Missings: FALSE
- Target: “type”

**Source**


Donor: Stefan Aeberhard, email: stefan@coral.cs.jcu.edu.au
mlr_tasks_zoo

References


See Also

Dictionary of Tasks: mlr_tasks
as.data.table(mlr_tasks) for a complete table of all (also dynamically created) Tasks.

Other Task: TaskClassif, TaskRegr, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing, mlr_tasks_breast_cancer, mlr_tasks_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_penguins, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_zoo, mlr_tasks

---

mlr_tasks_zoo  Zoo Classification Task

Description

A classification task for the mlbench::Zoo data set. Rownames are stored as variable ".rownames" with column role "name".

Format

R6::R6Class inheriting from TaskClassif.

Construction

mlr_tasks$get("zoo")

mtsk("zoo")

Meta Information

- Task type: "classif"
- Dimensions: 101x17
- Properties: "multiclass"
- Has Missings: FALSE
- Target: "type"
mlr_task_generators

See Also

Dictionary of Tasks: mlr_tasks
as.data.table(mlr_tasks) for a complete table of all (also dynamically created) Tasks.
Other Task: TaskClassif, TaskRegr, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing, mlr_tasks_breast_cancer, mlr_tasks_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_penguins, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks

mlr_task_generators  Dictionary of Task Generators

Description

A simple mlr3misc::Dictionary storing objects of class TaskGenerator. Each task generator has an
associated help page, see mlr_task_generators_{id}.
This dictionary can get populated with additional task generators by add-on packages.
For a more convenient way to retrieve and construct task generators, see tgen()/tgens().

Format

R6::R6Class object inheriting from mlr3misc::Dictionary.

Methods

See mlr3misc::Dictionary.

S3 methods

• as.data.table(dict)
  mlr3misc::Dictionary -> data.table::data.table()
  Returns a data.table::data.table() with fields "key" and "packages" as columns.

See Also

Sugar functions: tgen(), tgens()
Other Dictionary: mlr_learners, mlr_measures, mlr_resamplings, mlr_tasks
Other TaskGenerator: TaskGenerator, mlr_task_generators_2dnormals, mlr_task_generators_cassini, mlr_task_generators_circle, mlr_task_generators_friedman1, mlr_task_generators_moons, mlr_task_generators_simplex, mlr_task_generators_smiley, mlr_task_generators_spirals, mlr_task_generators_xor

Examples

mlr_task_generators$get("smiley")
tgen("2dnormals")
Description

A TaskGenerator for the 2d normals task in `mlbench::mlbench.2dnormals()`.

Dictionary

This TaskGenerator can be instantiated via the dictionary `mlr_task_generators` or with the associated sugar function `tgen()`:

```
mlr_task_generators$get("2dnormals")
tgen("2dnormals")
```

Super class

`mlr3::TaskGenerator` -> `TaskGenerator2DNormals`

Methods

Public methods:

- `TaskGenerator2DNormals$new()`
- `TaskGenerator2DNormals$plot()`
- `TaskGenerator2DNormals$clone()`

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

Usage:

```
TaskGenerator2DNormals$new()
```

Method `plot()`: Creates a simple plot of generated data.

Usage:

```
TaskGenerator2DNormals$plot(n = 200L, pch = 19L, ...)
```

Arguments:

- `n` (integer(1))
  - Number of samples to draw for the plot. Default is 200.
- `pch` (integer(1))
  - Point char. Passed to `plot()`.
- `...` (any)
  - Additional arguments passed to `plot()`.

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

Usage:

```
TaskGenerator2DNormals$clone(deep = FALSE)
```

Arguments:

- `deep` Whether to make a deep clone.
See Also

Dictionary of TaskGenerators: mlr_task_generators
as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) TaskGenerator implementations.

Other TaskGenerator: TaskGenerator, mlr_task_generators_cassini, mlr_task_generators_circle, mlr_task_generators_friedman1, mlr_task_generators_moons, mlr_task_generators_simplex, mlr_task_generators_smiley, mlr_task_generators_spirals, mlr_task_generators_xor, mlr_task_generators

Examples

generator = tgen("2dnormals")
plot(generator, n = 200)

task = generator$generate(200)
str(task$data())

Cassini Classification Task Generator

Description

A TaskGenerator for the cassini task in mlbench::mlbench.cassini().

Dictionary

This TaskGenerator can be instantiated via the dictionary mlr_task_generators or with the associated sugar function tgen():

mlr_task_generators$get("cassini")
tgen("cassini")

Super class

mlr3::TaskGenerator -> TaskGeneratorCassini

Methods

Public methods:

• TaskGeneratorCassini$new()
• TaskGeneratorCassini$plot()
• TaskGeneratorCassini$clone()

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

Usage:
TaskGeneratorCassini$new()

**Method** plot(): Creates a simple plot of generated data.

*Usage:*
TaskGeneratorCassini$plot(n = 200L, pch = 19L, ...)

*Arguments:*

- **n** (integer(1))
  Number of samples to draw for the plot. Default is 200.
- **pch** (integer(1))
  Point char. Passed to `plot()`.
- **...** (any)
  Additional arguments passed to `plot()`.

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

*Usage:*
TaskGeneratorCassini$clone(deep = FALSE)

*Arguments:*

- **deep**
  Whether to make a deep clone.

**See Also**

- Dictionary of TaskGenerators: `mlr_task_generators`
- `as.data.table(mlr_resamplings)` for a complete table of all (also dynamically created) TaskGenerator implementations.
- Other TaskGenerator: `TaskGenerator, mlr_task_generators_2dnormals, mlr_task_generators_circle, mlr_task_generators_friedman1, mlr_task_generators_moons, mlr_task_generators_simplex, mlr_task_generators_smiley, mlr_task_generators_spirals, mlr_task_generators_xor, mlr_task_generators`

**Examples**

```r
generator = tgen("cassini")
plot(generator, n = 200)

task = generator$generate(200)
str(task$data())
```

---

**mlr_task_generators_circle**

*Circle Classification Task Generator*

**Description**

A TaskGenerator for the circle binary classification task in `mlbench::mlbench.circle()`. Creates a large circle containing a smaller circle.
Dictionary

This TaskGenerator can be instantiated via the dictionary `mlr_task_generators` or with the associated sugar function `tgen()`:

```r
mlr_task_generators$get("circle")
tgen("circle")
```

Super class

```
mlr3::TaskGenerator -> TaskGeneratorCircle
```

Methods

**Public methods:**

- `TaskGeneratorCircle$new()`
- `TaskGeneratorCircle$plot()`
- `TaskGeneratorCircle$clone()`

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

`Usage`

```r
TaskGeneratorCircle$new()
```

**Method `plot()`**: Creates a simple plot of generated data.

`Usage`

```r
TaskGeneratorCircle$plot(n = 200L, pch = 19L, ...)
```

**Arguments**

- `n` (integer(1))
  - Number of samples to draw for the plot. Default is 200.
- `pch` (integer(1))
  - Point char. Passed to `plot()`.
- `...` (any)
  - Additional arguments passed to `plot()`.

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

`Usage`

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

**Arguments**

- `deep` Whether to make a deep clone.

See Also

- Dictionary of TaskGenerators: `mlr_task_generators`
- `as.data.table(mlr_resamplings)` for a complete table of all (also dynamically created) TaskGenerator implementations.

Other TaskGenerator: `TaskGenerator, mlr_task_generators_2dnormals, mlr_task_generators_cassini, mlr_task_generators_friedman1, mlr_task_generators_moons, mlr_task_generators_simplex, mlr_task_generators_smiley, mlr_task_generators_spirals, mlr_task_generators_xor, mlr_task_generators`
Examples

generator = tgen("circle")
plot(generator, n = 200)

task = generator$generate(200)
str(task$data())

mlr_task_generators_friedman1

Friedman1 Regression Task Generator

Description

A TaskGenerator for the friedman1 task in mlbench::mlbench.friedman1().

Dictionary

This TaskGenerator can be instantiated via the dictionary mlr_task_generators or with the associated sugar function tgen():

mlr_task_generators$get("friedman1")
tgen("friedman1")

Super class

mlr3::TaskGenerator -> TaskGeneratorFriedman1

Methods

Public methods:

• TaskGeneratorFriedman1$new()
• TaskGeneratorFriedman1$clone()

Method new(): Creates a new instance of this R6 class.
Usage:
TaskGeneratorFriedman1$new()

Method clone(): The objects of this class are cloneable with this method.
Usage:
TaskGeneratorFriedman1$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.
See Also

Dictionary of TaskGenerators: mlr_task_generators
as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) TaskGenerator implementations.

Other TaskGenerator: TaskGenerator, mlr_task_generators_2dnormals, mlr_task_generators_cassini, mlr_task_generators_circle, mlr_task_generators_moons, mlr_task_generators_simplex, mlr_task_generators_smiley, mlr_task_generators_spirals, mlr_task_generators_xor, mlr_task_generators

Examples

generator = tgen("friedman1")
task = generator$generate(200)
str(task$data())

mlr_task_generators_moons

Moons Classification Task Generator

Description

A TaskGenerator creating two interleaving half circles ("moons") as binary classification problem.

Dictionary

This TaskGenerator can be instantiated via the dictionary mlr_task_generators or with the associated sugar function tgen():

mlr_task_generators$get("moons")
tgen("moons")

Super class

mlr3::TaskGenerator -> TaskGeneratorMoons

Methods

Public methods:

- TaskGeneratorMoons$new()
- TaskGeneratorMoons$plot()
- TaskGeneratorMoons$clone()

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

Usage:
TaskGeneratorMoons$new()
Method `plot()`: Creates a simple plot of generated data.

Usage:
```
TaskGeneratorMoons$plot(n = 200L, pch = 19L, ...)
```

Arguments:
- `n` (integer(1))
  - Number of samples to draw for the plot. Default is 200.
- `pch` (integer(1))
  - Point char. Passed to `plot()`.
- `...` (any)
  - Additional arguments passed to `plot()`.

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

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

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

See Also
- Dictionary of TaskGenerators: `mlr_task_generators`
- `as.data.table(mlr_resamplings)` for a complete table of all (also dynamically created) TaskGenerator implementations.
- Other TaskGenerator: `TaskGenerator, mlr_task_generators_2dnormals, mlr_task_generators_cassini, mlr_task_generators_circle, mlr_task_generators_friedman1, mlr_task_generators_simplex, mlr_task_generators_smiley, mlr_task_generators_spirals, mlr_task_generators_xor, mlr_task_generators`

Examples
```
generator = tgen("moons")
plot(generator, n = 200)

task = generator$generate(200)
str(task$data())
```

---

`mlr_task_generators_simplex`

*Simplex Classification Task Generator*

Description

A TaskGenerator for the simplex task in `mlbench::mlbench.simplex()`.

Note that the generator implemented in `mlbench` returns fewer samples than requested.
Dictionary

This TaskGenerator can be instantiated via the dictionary `mlr_task_generators` or with the associated sugar function `tgen()`:

```r
mlr_task_generators$getattr("simplex")
tgen("simplex")
```

Super class

```r
mlr3::TaskGenerator -> TaskGeneratorSimplex
```

Methods

Public methods:

- `TaskGeneratorSimplex$new()`
- `TaskGeneratorSimplex$plot()`
- `TaskGeneratorSimplex$clone()`

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

Usage:

```r
TaskGeneratorSimplex$new()
```

Method `plot()`: Creates a simple plot of generated data.

Usage:

```r
TaskGeneratorSimplex$plot(n = 200L, pch = 19L, ...)
```

Arguments:

- `n` (integer(1))
  - Number of samples to draw for the plot. Default is 200.
- `pch` (integer(1))
  - Point char. Passed to `plot()`.
- `...` (any)
  - Additional arguments passed to `plot()`.

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

Usage:

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

Arguments:

- `deep` Whether to make a deep clone.

See Also

Dictionary of TaskGenerators: `mlr_task_generators`

as.data.table(`mlr_resamplings`) for a complete table of all (also dynamically created) TaskGenerator implementations.

Other TaskGenerator: `TaskGenerator, mlr_task_generators_2dnormals, mlr_task_generators_cassini, mlr_task_generators_circle, mlr_task_generators_friedman1, mlr_task_generators_moons, mlr_task_generators_smiley, mlr_task_generators_spirals, mlr_task_generators_xor, mlr_task_generators`
Examples

```r
generator = tgen("simplex")
plot(generator, n = 200)

task = generator$generate(200)
str(task$data())
```

---

**Description**

A TaskGenerator for the smiley task in `mlbench::mlbench.smiley()`.

**Dictionary**

This TaskGenerator can be instantiated via the dictionary `mlr_task_generators` or with the associated sugar function `tgen()`:

```r
class(task_generator_smiley)
tgen("smiley")
```

**Super class**

`mlr3::TaskGenerator` -> TaskGeneratorSmiley

**Methods**

- Public methods:

  ```r
  TaskGeneratorSmiley$new()
  TaskGeneratorSmiley$plot()
  TaskGeneratorSmiley$clone()
  ```

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

- **Method plot()**: Creates a simple plot of generated data.
  
  **Usage**:
  
  ```r
  TaskGeneratorSmiley$plot(n = 200L, pch = 19L, ...)
  ```

  **Arguments**:
  
  ```r
  n (integer(1))
  Number of samples to draw for the plot. Default is 200.
  pch (integer(1))
  Point char. Passed to plot().
  ```
Method clone(): The objects of this class are cloneable with this method.

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

Arguments:
- deep Whether to make a deep clone.

See Also
Dictionary of TaskGenerators: `mlr_task_generators`

as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) TaskGenerator implementations.

Other TaskGenerator: TaskGenerator, mlr_task_generators_2dnormals, mlr_task_generators_cassini, mlr_task_generators_circle, mlr_task_generators_friedman1, mlr_task_generators_moons, mlr_task_generators_simplex, mlr_task_generators_spirals, mlr_task_generators_xor, mlr_task_generators

Examples
```
generator = tgen("smiley")
plot(generator, n = 200)

task = generator$generate(200)
str(task$data())
```
Methods

Public methods:

• TaskGeneratorSpirals$new()
• TaskGeneratorSpirals$plot()
• TaskGeneratorSpirals$clone()

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

Usage:
TaskGeneratorSpirals$new()

Method plot(): Creates a simple plot of generated data.

Usage:
TaskGeneratorSpirals$plot(n = 200L, pch = 19L, ...)

Arguments:

n (integer(1))
Number of samples to draw for the plot. Default is 200.
pch (integer(1))
Point char. Passed to plot().
...
(any)
Additional arguments passed to plot().

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

Usage:
TaskGeneratorSpirals$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

See Also

Dictionary of TaskGenerators: mlr_task_generators
as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) TaskGenerator implementations.

Other TaskGenerator: TaskGenerator, mlr_task_generators_2dnormals, mlr_task_generators_cassini, mlr_task_generators_circle, mlr_task_generators_friedman1, mlr_task_generators_moons, mlr_task_generators_simplex, mlr_task_generators_smiley, mlr_task_generators_xor, mlr_task_generators

Examples

generator = tgen("spirals")
plot(generator, n = 200)

task = generator$generate(200)
str(task$data())
**Description**

A TaskGenerator for the xor task in `mlbench::mlbench.xor()`.

**Dictionary**

This TaskGenerator can be instantiated via the dictionary `mlr_task_generators` or with the associated sugar function `tgen()`:

```r
tgen("xor")
mlr_task_generators$get("xor")
```

**Super class**

`mlr3::TaskGenerator` -> `TaskGeneratorXor`

**Methods**

**Public methods:**

- `TaskGeneratorXor$new()`
- `TaskGeneratorXor$plot()`
- `TaskGeneratorXor$clone()`

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

*Usage:*

```r
TaskGeneratorXor$new()
```

**Method plot():** Creates a simple plot of generated data.

*Usage:*

```r
TaskGeneratorXor$plot(n = 200L, pch = 19L, ...)
```

*Arguments:*

- `n` (`integer(1)`)
  
  Number of samples to draw for the plot. Default is 200.

- `pch` (`integer(1)`)
  
  Point char. Passed to `plot()`.

- `...` (`any`)
  
  Additional arguments passed to `plot()`.

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

*Usage:*

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

*Arguments:*

- `deep` Whether to make a deep clone.
predict.Learner

**Predict Method for Learners**

**Description**

Extends the generic `stats::predict()` with a method for Learner. Note that this function is intended as glue code to be used in third party packages. We recommend to work with the Learner directly, i.e. calling `learner$predict()` or `learner$predict_newdata()` directly.

Performs the following steps:

- Sets additional hyperparameters passed to this function.
- Creates a Prediction object by calling `learner$predict_newdata()`.
- Returns (subset of) Prediction.

**Usage**

```r
## S3 method for class 'Learner'
predict(object, newdata, predict_type = NULL, ...)
```

**Arguments**

- `object` *(Learner)*
  
  Any Learner.

- `newdata` *(data.frame())*
  
  New data to predict on.

- `predict_type` *(character(1))*
  
  The predict type to return. Set to `<Prediction>` to retrieve the complete Prediction object. If set to NULL (default), the first predict type for the respective class of the Learner as stored in `mlr_reflections` is used.

- `...` *(any)*
  
  Hyperparameters to pass down to the Learner.

**See Also**

Dictionary of TaskGenerators: `mlr_task_generators` as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) TaskGenerator implementations.

Other TaskGenerator: `TaskGenerator`, `mlr_task_generators_2dnormals`, `mlr_task_generators_cassini`, `mlr_task_generators_circle`, `mlr_task_generators_friedman1`, `mlr_task_generators_moons`, `mlr_task_generators_simplex`, `mlr_task_generators_smiley`, `mlr_task_generators_spirals`, `mlr_task_generators`

**Examples**

```r
generator = tgen("xor")
plot(generator, n = 200)

task = generator$generate(200)
str(task$data())
```
Examples

task = tsk("spam")

learner = lrn("classif.rpart", predict_type = "prob")
learner$train(task)
predict(learner, task$data(1:3), predict_type = "response")
predict(learner, task$data(1:3), predict_type = "prob")
predict(learner, task$data(1:3), predict_type = "<Prediction>")

Abstract Prediction Object

Description

This is the abstract base class for task objects like PredictionClassif or PredictionRegr.

Prediction objects store the following information:

1. The row ids of the test set
2. The corresponding true (observed) response.
3. The corresponding predicted response.
4. Additional predictions based on the class and predict_type. E.g., the class probabilities for classification or the estimated standard error for regression.

Note that this object is usually constructed via a derived classes, e.g. PredictionClassif or PredictionRegr.

S3 Methods

- as.data.table(rr)
  Prediction -> data.table::data.table()
  Converts the data to a data.table::data.table().
- c(..., keep_duplicates = TRUE)
  (Prediction, Prediction, ...) -> Prediction
  Combines multiple Predictions to a single Prediction. If keep_duplicates is FALSE and there are duplicated row ids, the data of the former passed objects get overwritten by the data of the later passed objects.

Public fields

data (named list())
  Internal data structure.

task_type (character(1))
  Required type of the Task.

task_properties (character())
  Required properties of the Task.
predict_types (character())
   Set of predict types this object stores.

man (character(1))
   String in the format [pkg]:[topic] pointing to a manual page for this object. Defaults to NA, but can be set by child classes.

Active bindings

row_ids (integer())
   Vector of row ids for which predictions are stored.

truth (any)
   True (observed) outcome.

missing (integer())
   Returns row_ids for which the predictions are missing or incomplete.

Methods

Public methods:
• Prediction$format()
• Prediction$print()
• Prediction$help()
• Prediction$score()
• Prediction$clone()

Method format(): Helper for print outputs.
   Usage:
   Prediction$format()

Method print(): Printer.
   Usage:
   Prediction$print(...)

Arguments:
   ... (ignored).

Method help(): Opens the corresponding help page referenced by field $man.
   Usage:
   Prediction$help()

Method score(): Calculates the performance for all provided measures Task and Learner may be NULL for most measures, but some measures need to extract information from these objects. Note that the predict_sets of the measures are ignored by this method, instead all predictions are used.
   Usage:
**PredictionClassif**

**Prediction$score**(
  measures = NULL,
  task = NULL,
  learner = NULL,
  train_set = NULL
)

*Arguments:*

- `measures` *(Measure | list of Measure)*
  Measure(s) to calculate.
- `task` *(Task)*
- `learner` *(Learner)*
- `train_set` *(integer())*

*Returns:* Prediction.

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

**Usage:**

`Prediction$clone(deep = FALSE)`

*Arguments:*

- `deep` Whether to make a deep clone.

**See Also**

Other Prediction: PredictionClassif, PredictionRegr

---

**Description**

This object wraps the predictions returned by a learner of class LearnerClassif, i.e. the predicted response and class probabilities.

If the response is not provided during construction, but class probabilities are, the response is calculated from the probabilities: the class label with the highest probability is chosen. In case of ties, a label is selected randomly.

**Thresholding**

If probabilities are stored, it is possible to change the threshold which determines the predicted class label. Usually, the label of the class with the highest predicted probability is selected. For binary classification problems, such an threshold defaults to 0.5. For cost-sensitive or imbalanced classification problems, manually adjusting the threshold can increase the predictive performance.

- For binary problems only a single threshold value can be set. If the probability exceeds the threshold, the positive class is predicted. If the probability equals the threshold, the label is selected randomly.
• For binary and multi-class problems, a named numeric vector of thresholds can be set. The length and names must correspond to the number of classes and class names, respectively. To determine the class label, the probabilities are divided by the threshold. This results in a ratio > 1 if the probability exceeds the threshold, and a ratio < 1 otherwise. Note that it is possible that either none or multiple ratios are greater than 1 at the same time. Anyway, the class label with maximum ratio is selected. In case of ties in the ratio, one of the tied class labels is selected randomly.

Note that there are the following edge cases for threshold equal to 0 which are handled specially:

1. With threshold 0 the resulting ratio gets Inf and thus gets always selected. If there are multiple ratios with value Inf, one is selected according to ties_method (randomly per default).
2. If additionally the predicted probability is also 0, the ratio 0/0 results in NaN values. These are simply replaced by 0 and thus will never get selected.

**Super class**

mlr3::Prediction -> PredictionClassif

**Active bindings**

`response (factor())`
Access to the stored predicted class labels.

`prob (matrix())`
Access to the stored probabilities.

`confusion (matrix())`
Confusion matrix, as resulting from the comparison of truth and response. Truth is in columns, predicted response is in rows.

**Methods**

**Public methods:**

- `PredictionClassif$new()
- `PredictionClassif$set_threshold()

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

*Usage:*

PredictionClassif$new(
  task = NULL,
  row_ids = task$row_ids,
  truth = task$truth(),
  response = NULL,
  prob = NULL,
  check = TRUE
)

*Arguments:*

PredictionClassif

task (TaskClassif)

Task, used to extract defaults for row_ids and truth.

row_ids (integer())

Row ids of the predicted observations, i.e. the row ids of the test set.

truth (factor())

True (observed) labels. See the note on manual construction.

response (character() | factor())

Vector of predicted class labels. One element for each observation in the test set. Character vectors are automatically converted to factors. See the note on manual construction.

prob (matrix())

Numeric matrix of posterior class probabilities with one column for each class and one row for each observation in the test set. Columns must be named with class labels, row names are automatically removed. If prob is provided, but response is not, the class labels are calculated from the probabilities using max.col() with ties.method set to "random".

check (logical(1))

If TRUE, performs some argument checks and predict type conversions.

Method set_threshold(): Sets the prediction response based on the provided threshold. See the section on thresholding for more information.

Usage:
PredictionClassif$set_threshold(threshold, ties_method = "random")

Arguments:
threshold (numeric()).

ties_method (character(1))

One of "random", "first" or "last" (c.f. max.col()) to determine how to deal with tied probabilities.

Returns: Returns the object itself, but modified by reference. You need to explicitly $clone() the object beforehand if you want to keeps the object in its previous state.

Note

If this object is constructed manually, make sure that the factor levels for truth have the same levels as the task, in the same order. In case of binary classification tasks, the positive class label must be the first level.

See Also

Other Prediction: PredictionRegr, Prediction

Examples

task = tsk("penguins")
learner = lrn("classif.rpart", predict_type = "prob")
learner$train(task)
p = learner$predict(task)
p$predict_types
head(as.data.table(p))
# confusion matrix
p$confusion

# change threshold
th = c(0.05, 0.9, 0.05)
names(th) = task$class_names

# new predictions
p$set_threshold(th)$response
p$score(measures = msr("classif.ce"))

---

**PredictionData**  
*Convert to PredictionData*

**Description**

Objects of type PredictionData serve as an intermediate representation for objects of type Prediction. It is an internal data structure, implemented to optimize runtime and solve some issues emerging while serializing R6 objects. End-users typically do not need to worry about the details, package developers are advised to continue reading for some technical information.

Unlike most other mlr3 objects, PredictionData relies on the S3 class system. The following operations must be supported to extend mlr3 for new task types:

- `as_prediction_data()` converts objects to class PredictionData, e.g. objects of type Prediction.
- `as_prediction()` converts objects to class Prediction, e.g. objects of type PredictionData.
- `check_prediction_data()` is called on the return value of the predict method of a Learner to perform assertions and type conversions. Returns an update object of class PredictionData.
- `is_missing_prediction_data()` is used for the fallback learner (see Learner) to impute missing predictions. Returns vector with row ids which need imputation.

**Usage**

```r
check_prediction_data(pdata)

is_missing_prediction_data(pdata)
```

```r
## S3 method for class 'PredictionDataClassif'
check_prediction_data(pdata)

## S3 method for class 'PredictionDataClassif'
is_missing_prediction_data(pdata)

## S3 method for class 'PredictionDataClassif'
c(..., keep_duplicates = TRUE)
```
## S3 method for class 'PredictionDataRegr'
check_prediction_data(pdata)

## S3 method for class 'PredictionDataRegr'
is_missing_prediction_data(pdata)

## S3 method for class 'PredictionDataRegr'
c(..., keep_duplicates = TRUE)

### Arguments

- **pdata** (PredictionData)
  Named list inheriting from "PredictionData".
- **...** (one or more PredictionData objects).
- **keep_duplicates** (logical(1)) If TRUE, the combined PredictionData object is filtered for duplicated row ids (starting from last).

---

**Description**

This object wraps the predictions returned by a learner of class LearnerRegr, i.e. the predicted response and standard error. Additionally, probability distributions implemented in distr6 are supported.

**Super class**

mlr3::Prediction -> PredictionRegr

**Active bindings**

- **response** (numeric())
  Access the stored predicted response.
- **se** (numeric())
  Access the stored standard error.
- **distr** (distr6::VectorDistribution)
  Access the stored vector distribution. Requires package distr6.

**Methods**

**Public methods:**

- **PredictionRegr$new()**

**Method** new(): Creates a new instance of this R6 class.
Usage:
PredictionRegr$new(
  task = NULL,
  row_ids = task$row_ids,
  truth = task$truth(),
  response = NULL,
  se = NULL,
  distr = NULL,
  check = TRUE
)

Arguments:
task (TaskRegr)
  Task, used to extract defaults for row_ids and truth.
row_ids (integer())
  Row ids of the predicted observations, i.e. the row ids of the test set.
truth (numeric())
  True (observed) response.
response (numeric())
  Vector of numeric response values. One element for each observation in the test set.
se (numeric())
  Numeric vector of predicted standard errors. One element for each observation in the test set.
distr (distr6::VectorDistribution)
  VectorDistribution from distr6. Each individual distribution in the vector represents the random variable 'survival time' for an individual observation.
check (logical(1))
  If TRUE, performs some argument checks and predict type conversions.

See Also
Other Prediction: PredictionClassif, Prediction

Examples
  task = tsk("boston_housing")
  learner = lrn("regr.featureless", predict_type = "se")
  p = learner$train(task)$predict(task)
  p$predict_types
  head(as.data.table(p))

resample

Resample a Learner on a Task

Description

Runs a resampling (possibly in parallel): Repeatedly apply Learner learner on a training set of Task task to train a model, then use the trained model to predict observations of a test set. Training and test sets are defined by the Resampling resampling.
resample

Usage

resample(
  task,
  learner,
  resampling,
  store_models = FALSE,
  store_backends = TRUE
)

Arguments

task (Task).
learner (Learner).
resampling (Resampling).
store_models (logical(1))
  Keep the fitted model after the test set has been predicted? Set to TRUE if you want to further analyse the models or want to extract information like variable importance.
store_backends (logical(1))
  Keep the DataBackend of the Task in the ResampleResult? Set to TRUE if your performance measures require a Task, or to analyse results more conveniently. Set to FALSE to reduce the file size and memory footprint after serialization. The current default is TRUE, but this eventually will be changed in a future release.

Value

ResampleResult.

Parallelization

This function can be parallelized with the future package. One job is one resampling iteration, and all jobs are send to an apply function from future.apply in a single batch. To select a parallel backend, use future::plan().

Progress Bars

This function supports progress bars via the package progressr. Simply wrap the function in progressr::with_progress() to enable them. We recommend to use package progress as backend; enable with progressr::handlers("progress").

Logging

The mlr3 uses the lgr package for logging. lgr supports multiple log levels which can be queried with getOption("lgr.log_levels").

To suppress output and reduce verbosity, you can lower the log from the default level "info" to "warn":

lgr::get_logger("mlr3")$set_threshold("warn")
To get additional log output for debugging, increase the log level to "debug" or "trace":

```r
dlgr::get_logger("mlr3")$set_threshold("debug")
```

To log to a file or a database, see the documentation of `lgr::lgr-package`.

**Note**

The fitted models are discarded after the predictions have been computed in order to reduce memory consumption. If you need access to the models for later analysis, set `store_models` to `TRUE`.

**Examples**

```r
# Explicitly instantiate the resampling for this task for reproducibility
set.seed(123)
resampling$instantiate(task)

rr = resample(task, learner, resampling)
print(rr)

# Retrieve performance
rr$score(msr("classif.ce"))
rr$aggregate(msr("classif.ce"))

# merged prediction objects of all resampling iterations
pred = rr$prediction()
pred$confusion

# Repeat resampling with featureless learner
rr_featureless = resample(task, lrn("classif.featureless"), resampling)

# Convert results to BenchmarkResult, then combine them
bmr1 = as_benchmark_result(rr)
bmr2 = as_benchmark_result(rr_featureless)
print(bmr1$combine(bmr2))
```

---

**Description**

This is the result container object returned by `resample()`.

Note that all stored objects are accessed by reference. Do not modify any object without cloning it first.
S3 Methods

- as.data.table(rr, reassemble_learners = TRUE, convert_predictions = TRUE, predict_sets = "test")
  ResampleResult -> data.table::data.table()
  Returns a tabular view of the internal data.

- c(...)
  (ResampleResult, ...) -> BenchmarkResult
  Combines multiple objects convertible to BenchmarkResult into a new BenchmarkResult.

Public fields

data (ResultData)
  Internal data storage object of type ResultData. We discourage users to directly work with
  this field. Use as.table.table(ResampleResult) instead.

view (character(1))
  Subset of uhashes in the ResultData object to operate on. This field is for internal optimiza-
  tions, i.e. to avoid unnecessary cloning.

Active bindings

task_type (character(1))
  Task type of objects in the ResampleResult, e.g. "classif" or "regr". This is NA for empty
  ResampleResults.

uhash (character(1))
  Unique hash for this object.

task (Task)
  The task resample() operated on.

learner (Learner)
  Learner prototype resample() operated on. For a list of trained learners, see methods $learn-
iners().

resampling (Resampling)
  Instantiated Resampling object which stores the splits into training and test.

learners (list of Learner)
  List of trained learners, sorted by resampling iteration.

warnings (data.table::data.table())
  A table with all warning messages. Column names are "iteration" and "msg". Note that
  there can be multiple rows per resampling iteration if multiple warnings have been recorded.

ersors (data.table::data.table())
  A table with all error messages. Column names are "iteration" and "msg". Note that there
  can be multiple rows per resampling iteration if multiple errors have been recorded.

Methods

Public methods:

- ResampleResult$new()
- ResampleResult$format()
- ResampleResult$print()
- ResampleResult$help()
- ResampleResult$prediction()
- ResampleResult$predictions()
- ResampleResult$score()
- ResampleResult$aggregate()
- ResampleResult$filter()
- ResampleResult$clone()

**Method new()**: Creates a new instance of this **R6** class. An alternative construction method is provided by `as_resample_result()`.

*Usage:*

```r
ResampleResult$new(data = ResultData$new(), view = NULL)
```

*Arguments:*

- `data` (**ResultData | data.table()**)
  - An object of type `ResultData`, either extracted from another `ResampleResult`, another `BenchmarkResult`, or manually constructed with `as_result_data()`.
- `view` (**character()**)
  - Single uhash of the `ResultData` to operate on. Used internally for optimizations.

**Method format()**: Helper for print outputs.

*Usage:*

```r
ResampleResult$format()
```

**Method print()**: Printer.

*Usage:*

```r
ResampleResult$print()
```

*Arguments:*

- `...` (ignored).

**Method help()**: Opens the corresponding help page referenced by field `$man`.

*Usage:*

```r
ResampleResult$help()
```

**Method prediction()**: Combined **Prediction** of all individual resampling iterations, and all provided predict sets. Note that performance measures do not operate on this object, but instead on each prediction object separately and then combine the performance scores with the aggregate function of the respective `Measure`.

*Usage:*

```r
ResampleResult$prediction(predict_sets = "test")
```

*Arguments:*

- `predict_sets` (**character()**)

Returns: Prediction. Subset of {"train", "test"}.

Method predictions(): List of prediction objects, sorted by resampling iteration. If multiple sets are given, these are combined to a single one for each iteration.

Usage:
ResampleResult$predictions(predict_sets = "test")

Arguments:
predict_sets (character())
  Subset of {"train", "test"}.

Returns: List of Prediction objects, one per element in predict_sets.

Method score(): Returns a table with one row for each resampling iteration, including all involved objects: Task, Learner, Resampling, iteration number (integer(1)), and Prediction. Additionally, a column with the individual (per resampling iteration) performance is added for each Measure in measures, named with the id of the respective measure id. If measures is NULL, measures defaults to the return value of default_measures().

Usage:
ResampleResult$score(
  measures = NULL,
  ids = TRUE,
  conditions = FALSE,
  predict_sets = "test"
)

Arguments:
measures (Measure | list of Measure)
  Measure(s) to calculate.
ids (logical(1))
  If ids is TRUE, extra columns with the ids of objects ("task_id", "learner_id", "resampling_id") are added to the returned table. These allow to subset more conveniently.
conditions (logical(1))
  Adds condition messages ("warnings", "errors") as extra list columns of character vectors to the returned table
predict_sets (character())
  Vector of predict sets (\{"train", "test"\}) to construct the Prediction objects from. Default is "test".

Returns: data.table::data.table().

Method aggregate(): Calculates and aggregates performance values for all provided measures, according to the respective aggregation function in Measure. If measures is NULL, measures defaults to the return value of default_measures().

Usage:
ResampleResult$aggregate(measures = NULL)

Arguments:
measures (Measure | list of Measure)
  Measure(s) to calculate.
Returns: Named numeric().

Method filter(): Subsets the ResampleResult, reducing it to only keep the iterations specified in iters.

Usage:
ResampleResult$filter(iters)

Arguments:
iters (integer())
  Resampling iterations to keep.

Returns: Returns the object itself, but modified by reference. You need to explicitly $clone() the object beforehand if you want to keep the object in its previous state.

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

Usage:
ResampleResult$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

Examples

task = tsk("penguins")
learner = lrn("classif.rpart")
resampling = rmp("cv", folds = 3)
rr = resample(task, learner, resampling)
print(rr)

rr$aggregate(msr("classif.acc"))
rr$prediction()
rr$prediction()$confusion
rr$warnings
rr$errors

<table>
<thead>
<tr>
<th>Resampling</th>
<th>Resampling Class</th>
</tr>
</thead>
</table>

Description

This is the abstract base class for resampling objects like ResamplingCV and ResamplingBootstrap. The objects of this class define how a task is partitioned for resampling (e.g., in resample() or benchmark()), using a set of hyperparameters such as the number of folds in cross-validation. Resampling objects can be instantiated on a Task, which applies the strategy on the task and manifests in a fixed partition of row_ids of the Task.

Predefined resamplings are stored in the dictionary mlr_resamplings, e.g. cv or bootstrap.
Stratification

All derived classes support stratified sampling. The stratification variables are assumed to be discrete and must be stored in the Task with column role "stratum". In case of multiple stratification variables, each combination of the values of the stratification variables forms a strata.

First, the observations are divided into subpopulations based on one or multiple stratification variables (assumed to be discrete), c.f. task$strata.

Second, the sampling is performed in each of the k subpopulations separately. Each subgroup is divided into iter training sets and iter test sets by the derived Resampling. These sets are merged based on their iteration number: all training sets from all subpopulations with iteration 1 are combined, then all training sets with iteration 2, and so on. Same is done for all test sets. The merged sets can be accessed via $train_set(i) and $test_set(i), respectively.

Grouping / Blocking

All derived classes support grouping of observations. The grouping variable is assumed to be discrete and must be stored in the Task with column role "group".

Observations in the same group are treated like a "block" of observations which must be kept together. These observations either all go together into the training set or together into the test set.

The sampling is performed by the derived Resampling on the grouping variable. Next, the grouping information is replaced with the respective row ids to generate training and test sets. The sets can be accessed via $train_set(i) and $test_set(i), respectively.

Public fields

id (character(1))
Identifier of the object. Used in tables, plot and text output.

param_set (paradox::ParamSet)
Set of hyperparameters.

instance (any)
During instantiate(), the instance is stored in this slot in an arbitrary format. Note that if a grouping variable is present in the Task, a Resampling may operate on the group ids internally instead of the row ids (which may lead to confusion).
It is advised to not work directly with the instance, but instead only use the getters $train_set() and $test_set().

task_hash (character(1))
The hash of the Task which was passed to r$instantiate().

task_nrow (integer(1))
The number of observations of the Task which was passed to r$instantiate().

duplicated_ids (logical(1))
If TRUE, duplicated rows can occur within a single training set or within a single test set. E.g., this is TRUE for Bootstrap, and FALSE for cross-validation. Only used internally.

man (character(1))
String in the format [pkg]:[topic] pointing to a manual page for this object. Defaults to NA, but can be set by child classes.
Active bindings

is_instantiated (logical(1))
Is TRUE if the resampling has been instantiated.

hash (character(1))
Hash (unique identifier) for this object.

Methods

Public methods:

• Resampling$new()
• Resampling$format()
• Resampling$print()
• Resampling$help()
• Resampling$instantiate()
• Resampling$train_set()
• Resampling$test_set()
• Resampling$clone()

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

Usage:
Resampling$new(
  id,
  param_set = ParamSet$new(),
  duplicated_ids = FALSE,
  man = NA_character_
)

Arguments:

id (character(1))
  Identifier for the new instance.

param_set (paradox::ParamSet)
  Set of hyperparameters.

duplicated_ids (logical(1))
  Set to TRUE if this resampling strategy may have duplicated row ids in a single training set or test set.
  Note that this object is typically constructed via a derived classes, e.g. ResamplingCV or ResamplingHoldout.

man (character(1))
  String in the format [pkg]::[topic] pointing to a manual page for this object. The referenced help package can be opened via method $help().

Method format(): Helper for print outputs.

Usage:
Resampling$format()

Method print(): Printer.
Usage:
Resampling$print(...)

Arguments:
... (ignored).

Method help(): Opens the corresponding help page referenced by field $man.

Usage:
Resampling$help()

Method instantiate(): Materializes fixed training and test splits for a given task and stores them in \( r\$instance \) in an arbitrary format.

Usage:
Resampling$instantiate(task)

Arguments:
task (Task)
  Task used for instantiation.

Returns: Returns the object itself, but modified by reference. You need to explicitly $clone() the object beforehand if you want to keeps the object in its previous state.

Method train_set(): Returns the row ids of the i-th training set.

Usage:
Resampling$train_set(i)

Arguments:
i (integer(1))
  Iteration.

Returns: (integer()) of row ids.

Method test_set(): Returns the row ids of the i-th test set.

Usage:
Resampling$test_set(i)

Arguments:
i (integer(1))
  Iteration.

Returns: (integer()) of row ids.

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

Usage:
Resampling$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.
See Also

Dictionary of Resamplings: mlr_resamplings

as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) Resampling implementations.

Other Resampling: mlr_resamplings_bootstrap, mlr_resamplings_custom, mlr_resamplings_cv, mlr_resamplings_holdout, mlr_resamplings_insampling, mlr_resamplings_loo, mlr_resamplings_repeated_cv, mlr_resamplings_subsampling, mlr_resamplings

Other Resampling: mlr_resamplings_bootstrap, mlr_resamplings_custom, mlr_resamplings_cv, mlr_resamplings_holdout, mlr_resamplings_insampling, mlr_resamplings_loo, mlr_resamplings_repeated_cv, mlr_resamplings_subsampling, mlr_resamplings

Examples

r = rsmp("subsampling")

# Default parametrization
r$param_set$values

# Do only 3 repeats on 10% of the data
r$param_set$values = list(ratio = 0.1, repeats = 3)

# Instantiate on penguins task

 task = tsk("penguins")
r$instantiate(task)

# Extract train/test sets

 train_set = r$train_set(1)
print(train_set)
intersect(train_set, r$test_set(1))

# Another example: 10-fold CV

r = rsmp("cv")$instantiate(task)
r$train_set(1)

# Stratification

 task = tsk("pima")
prop.table(table(task$truth())) # moderately unbalanced
task$col_roles$stratum = task$target_names

r = rsmp("subsampling")
r$instantiate(task)
prop.table(table(task$truth(r$train_set(1)))) # roughly same proportion

---

set_threads

Set the Number of Threads
Description

Control the parallelism via threading while calling external packages from mlr3.

For example, the random forest implementation in package ranger (connected via mlr3learners) supports threading via OpenMP. The number of threads to use can be set via hyperparameter num.threads, and defaults to 1. By calling set_threads(x, 4) with x being a ranger learner, the hyperparameter is changed so that 4 cores are used.

If the object x does not support threading, x is returned as-is. If applied to a list, recurses through all list elements.

Note that threading is incompatible with other parallelization techniques such as forking via the future::plan future::multicore. For this reason all learners connected to mlr3 have threading disabled in their defaults.

Usage

```r
set_threads(x, n = availableCores())
## Default S3 method:
set_threads(x, n = availableCores())
## S3 method for class 'Learner'
set_threads(x, n = availableCores())
## S3 method for class 'list'
set_threads(x, n = availableCores())
```

Arguments

- `x` (any)
  
  Object to set threads for, e.g. a Learner. This object is modified in-place.

- `n` (integer(1))
  
  Number of threads to use. Defaults to `parallelly::availableCores()`.

Value

Same object as input x (changed in-place), with possibly updated parameter values.

---

<table>
<thead>
<tr>
<th>Task</th>
<th>Task Class</th>
</tr>
</thead>
</table>

Description

This is the abstract base class for TaskSupervised and TaskUnsupervised. TaskClassif and TaskRegr inherit from TaskSupervised. More supervised tasks are implemented in mlr3proba, unsupervised cluster tasks in package mlr3cluster.

Tasks serve two purposes:
1. Tasks wrap a \texttt{DataBackend}, an object to transparently interface different data storage types.

2. Tasks store meta-information, such as the role of the individual columns in the \texttt{DataBackend}. For example, for a classification task a single column must be marked as target column, and others as features.

Predefined (toy) tasks are stored in the dictionary \texttt{mlr_tasks}, e.g. \texttt{penguins} or \texttt{boston_housing}. More toy tasks can be found in the dictionary after loading \texttt{mlr3data}.

\textbf{S3 methods}

- \texttt{as.data.table(t)}
  
  \texttt{Task} \rightarrow \texttt{data.table::data.table()}
  
  Returns the complete data as \texttt{data.table::data.table()}.

\textbf{Task mutators}

The following methods change the task in-place:

- Any modification of the lists $\text{col\_roles}$ or $\text{row\_roles}$. This provides a different "view" on the data without altering the data itself.
- Modification of column or row roles via $\text{set\_col\_roles()}$ or $\text{set\_row\_roles()}$, respectively.
- $\text{filter()}$ and $\text{select()}$ subset the set of active rows or features in $\text{row\_roles}$ or $\text{col\_roles}$, respectively. This provides a different "view" on the data without altering the data itself.
- $\text{rbind()}$ and $\text{cbind()}$ change the task in-place by binding rows or columns to the data, but without modifying the original \texttt{DataBackend}. Instead, the methods first create a new \texttt{DataBackendDataTable} from the provided new data, and then merge both backends into an abstract \texttt{DataBackend} which merges the results on-demand.
- $\text{rename()}$ wraps the \texttt{DataBackend} of the Task in an additional \texttt{DataBackend} which deals with the renaming. Also updates $\text{col\_roles}$ and $\text{col\_info}$.

\textbf{Public fields}

- \texttt{id (character(1))}
  
  Identifier of the object. Used in tables, plot and text output.

- \texttt{task\_type (character(1))}
  
  Task type, e.g. "classif" or "regr".
  
  For a complete list of possible task types (depending on the loaded packages), see \texttt{mlr_reflections$task\_types$type}.

- \texttt{backend (DataBackend)}
  
  Abstract interface to the data of the task.

- \texttt{col\_info (data.table::data.table())}
  
  Table with with 3 columns:
  
  - "id" (character()) stores the name of the column.
  - "type" (character()) holds the storage type of the variable, e.g. integer, numeric or character. See \texttt{mlr_reflections$task\_feature\_types} for a complete list of allowed types.
  - "levels" stores a vector of distinct values (levels) for ordered and unordered factor variables.
Task 189

man (character(1))
String in the format [pkg]::[topic] pointing to a manual page for this object. Defaults to NA, but can be set by child classes.

eextra_args (named list())
Additional arguments set during construction. Required for convert_task().

Active bindings

hash (character(1))
Hash (unique identifier) for this object.

row_ids (integer())
Returns the row ids of the DataBackend for observations with role "use".

row_names (data.table::data.table())
Returns a table with two columns:
• "row_id" (integer()), and
• "row_name" (character()).

feature_names (character())
Returns all column names with role == "feature".
Note that this vector determines the default order of columns for task$data(cols = NULL,...).
However, it is recommended to not rely on the order of columns, but instead always address columns by their name. The default order is not well defined after some operations, e.g. after task$cbind() or after processing via mlr3pipelines.

target_names (character())
Returns all column names with role "target".

properties (character())
Set of task properties. Possible properties are are stored in mlr_reflections$task_properties. The following properties are currently standardized and understood by tasks in mlr3:
• "strata": The task is resampled using one or more stratification variables (role "stratum").
• "groups": The task comes with grouping/blocking information (role "group").
• "weights": The task comes with observation weights (role "weight").

Note that above listed properties are calculated from the $col_roles and may not be set explicitly.

row_roles (named list())
Each row (observation) can have an arbitrary number of roles in the learning task:
• "use": Use in train / predict / resampling.
• "validation": Observations are hold back unless explicitly requested. Can be used as truly independent test set.

row_roles is a named list whose elements are named by row role and each element is an integer() vector of row ids. To alter the roles, just modify the list, e.g. with R's set functions (intersect(), setdiff(), union(), ...).

col_roles (named list())
Each column (feature) can have an arbitrary number of the following roles:
• "feature": Regular feature used in the model fitting process.
• "target": Target variable.
• "name": Row names / observation labels. To be used in plots. Can be queried with $row_names.
• "order": Data returned by $data() is ordered by this column (or these columns).
• "group": During resampling, observations with the same value of the variable with role "group" are marked as "belonging together". For each resampling iteration, observations of the same group will be exclusively assigned to be either in the training set or in the test set. Note that only up to one column may have this role.
• "stratum": Stratification variables. Multiple discrete columns may have this role.
• "weight": Observation weights. Only up to one column (assumed to be discrete) may have this role.
• "uri": URI pointing to an external resource, e.g., images on the file system.

col_roles is a named list whose elements are named by column role and each element is a character() vector of column names. To alter the roles, just modify the list, e.g. with R’s set functions (intersect(), setdiff(), union(), ...). The method $set_col_roles provides a convenient alternative to assign columns to roles.

nrow (integer(1))
Returns the total number of rows with role "use".

ncol (integer(1))
Returns the total number of columns with role "target" or "feature".

feature_types (data.table::data.table())
Returns a table with columns id and type where id are the column names of "active" features of the task and type is the storage type.

data Formats character()
Vector of supported data output formats. A specific format can be chosen in the $data() method.

strata (data.table::data.table())
If the task has columns designated with role "stratum", returns a table with one subpopulation per row and two columns:
• N(integer()) with the number of observations in the subpopulation, and
• row_id(list of integer()) as list column with the row ids in the respective subpopulation. Returns NULL if there are is no stratification variable. See Resampling for more information on stratification.

groups (data.table::data.table())
If the task has a column with designated role "group", a table with two columns:
• row_id(integer()), and
• grouping variable group(vector()).
Returns NULL if there are is no grouping column. See Resampling for more information on grouping.

order (data.table::data.table())
If the task has at least one column with designated role "order", a table with two columns:
• row_id(integer()), and
• ordering vector order(integer()).
Returns NULL if there are no order column.

weights (data.table::data.table())
If the task has a column with designated role "weight", a table with two columns:
  • row_id (integer()), and
  • observation weights weight (numeric()).
Returns NULL if there are no weight column.

uris (data.table::data.table())
If the task has a column with designated role "uri", a table with two columns:
  • row_id (integer()), and
  • uri (character()).
Returns NULL if there are no uri column.

Methods

Public methods:
• Task$new()
• Task$help()
• Task$format()
• Task$print()
• Task$data()
• Task$formula()
• Task$head()
• Task$levels()
• Task$missings()
• Task$filter()
• Task$select()
• Task$rbind()
• Task$cbind()
• Task$rename()
• Task$set_row_roles()
• Task$set_col_roles()
• Task$droplevels()
• Task$clone()

Method new(): Creates a new instance of this R6 class.
Note that this object is typically constructed via a derived classes, e.g. TaskClassif or TaskRegr.

Usage:
Task$new(id, task_type, backend, extra_args = list())

Arguments:
id (character(1))
  Identifier for the new instance.
task_type (character(1))
  Type of task, e.g. "regr" or "classif". Must be an element of mlr_reflections$task_types$type.
backend (DataBackend)
   Either a DataBackend, or any object which is convertible to a DataBackend with as_data_backend().
   E.g., a data.frame() will be converted to a DataBackendDataTable.
extra_args (named list())
   Named list of constructor arguments, required for converting task types via convert_task().

Method help(): Opens the corresponding help page referenced by field $man.
   Usage:
   Task$help()

Method format(): Helper for print outputs.
   Usage:
   Task$format()

Method print(): Printer.
   Usage:
   Task$print(...)
   Arguments:
   ... (ignored).

Method data(): Returns a slice of the data from the DataBackend in the data format specified
   by data_format. Rows default to observations with role "use", and columns default to features
   with roles "target" or "feature". If rows or cols are specified which do not exist in the
   DataBackend, an exception is raised.
   Rows and columns are returned in the order specified via the arguments rows and cols. If rows is
   NULL, rows are returned in the order of task$row_ids. If cols is NULL, the column order defaults
   to c(task$target_names, task$feature_names). Note that it is recommended to not rely on
   the order of columns, and instead always address columns with their respective column name.
   Usage:
   Task$data(rows = NULL, cols = NULL, data_format = "data.table", ordered = TRUE)
   Arguments:
   rows integer()
       Row indices.
   cols character()
       Column names.
   data_format (character(1))
       Desired data format, e.g. "data.table" or "Matrix".
   ordered (logical(1))
       If TRUE (default), data is ordered according to the columns with column role "order".
   Returns: Depending on the DataBackend, but usually a data.table::data.table().

Method formula(): Constructs a formula(), e.g. [target] ~ [feature_1] + [feature_2] + ... + [feature_k], using the features provided in argument rhs (defaults to all columns with role "feature", symbolized by ".").
   Usage:
Task

Task$formula(rhs = ".")

Arguments:

rhs (character(1))

Right hand side of the formula. Defaults to "." (all features of the task).

Returns: formula().

Method head(): Get the first n observations with role "use" of all columns with role "target" or "feature".

Usage:

Task$head(n = 6L)

Arguments:

n (integer(1)).

Returns: data.table::data.table() with n rows.

Method levels(): Returns the distinct values for columns referenced in cols with storage type "factor" or "ordered". Argument cols defaults to all such columns with role "target" or "feature".

Note that this function ignores the row roles, it returns all levels available in the DataBackend. To update the stored level information, e.g. after subsetting a task with $filter(), call $droplevels().

Usage:

Task$levels(cols = NULL)

Arguments:

cols character()

Column names.

Returns: named list().

Method missings(): Returns the number of missing observations for columns referenced in cols. Considers only active rows with row role "use". Argument cols defaults to all columns with role "target" or "feature".

Usage:

Task$missings(cols = NULL)

Arguments:

cols character()

Column names.

Returns: Named integer().

Method filter(): Subsets the task, keeping only the rows specified via row ids rows.

This operation mutates the task in-place. See the section on task mutators for more information.

Usage:

Task$filter(rows)

Arguments:

rows integer()

Row indices.
Returns: Returns the object itself, but modified by reference. You need to explicitly \$clone() the object beforehand if you want to keep the object in its previous state.

Method `select()`: Subsets the task, keeping only the features specified via column names `cols`. Note that you cannot deselect the target column, for obvious reasons.
This operation mutates the task in-place. See the section on task mutators for more information.

Usage:

```r
Task$select(cols)
```

Arguments:

`cols` character()
Column names.

Returns: Returns the object itself, but modified by reference. You need to explicitly \$clone() the object beforehand if you want to keep the object in its previous state.

Method `rbind()`: Adds additional rows to the `DataBackend` stored in `$backend`. New row ids are automatically created, unless `data` has a column whose name matches the primary key of the `DataBackend` (`task$backend$primary_key`). In case of name clashes of row ids, rows in `data` have higher precedence and virtually overwrite the rows in the `DataBackend`.
All columns with the roles "target", "feature", "weight", "group", "stratum", and "order" must be present in `data`. Columns only present in `data` but not in the `DataBackend` of `task` will be discarded.
This operation mutates the task in-place. See the section on task mutators for more information.

Usage:

```r
Task$rbind(data)
```

Arguments:

`data` (data.frame()).

Returns: Returns the object itself, but modified by reference. You need to explicitly \$clone() the object beforehand if you want to keep the object in its previous state.

Method `cbind()`: Adds additional columns to the `DataBackend` stored in `$backend`. The row ids must be provided as column in `data` (with column name matching the primary key name of the `DataBackend`). If this column is missing, it is assumed that the rows are exactly in the order of `$row_ids`. In case of name clashes of column names in `data` and `DataBackend`, columns in `data` have higher precedence and virtually overwrite the columns in the `DataBackend`.
This operation mutates the task in-place. See the section on task mutators for more information.

Usage:

```r
Task$cbind(data)
```

Arguments:

`data` (data.frame()).

Method `rename()`: Renames columns by mapping column names in `old` to new column names in `new` (element-wise).
This operation mutates the task in-place. See the section on task mutators for more information.
Task$rename(old, new)

Arguments:
old (character())
  Old names.
new (character())
  New names.

Returns: Returns the object itself, but modified by reference. You need to explicitly $clone() the object beforehand if you want to keep the object in its previous state.

Method set_row_roles(): Modifies the roles in $row_roles in-place.

Usage:
Task$set_row_roles(rows, roles = NULL, add_to = NULL, remove_from = NULL)

Arguments:
rows (integer())
  Row ids for which to change the roles for.
roles (character())
  Exclusively set rows to the specified roles (remove from other roles).
add_to (character())
  Add rows with row ids rows to roles specified in add_to. Rows keep their previous roles.
remove_from (character())
  Remove rows with row ids rows from roles specified in remove_from. Other row roles are preserved.

Details: Roles are first set exclusively (argument roles), then added (argument add_to) and finally removed (argument remove_from) from different roles.

Returns: Returns the object itself, but modified by reference. You need to explicitly $clone() the object beforehand if you want to keep the object in its previous state.

Method set_col_roles(): Modifies the roles in $col_roles in-place.

Usage:
Task$set_col_roles(cols, roles = NULL, add_to = NULL, remove_from = NULL)

Arguments:
cols (character())
  Column names for which to change the roles for.
roles (character())
  Exclusively set columns to the specified roles (remove from other roles).
add_to (character())
  Add columns with column names cols to roles specified in add_to. Columns keep their previous roles.
remove_from (character())
  Remove columns with column names cols from roles specified in remove_from. Other column roles are preserved.

Details: Roles are first set exclusively (argument roles), then added (argument add_to) and finally removed (argument remove_from) from different roles.
Returns: Returns the object itself, but modified by reference. You need to explicitly `clone()` the object beforehand if you want to keep the object in its previous state.

Method droplevels(): Updates the cache of stored factor levels, removing all levels not present in the current set of active rows. `cols` defaults to all columns with storage type "factor" or "ordered".

Usage:
Task$droplevels(cols = NULL)

Arguments:
cols character()
Column names.

Returns: Modified self.

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

Usage:
Task$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

See Also

Examples

# we use the inherited class TaskClassif here,
# Class Task is not intended for direct use
task = TaskClassif$new("penguins", palmerpenguins::penguins, target = "species")

task$nrow
task$ncol
task$feature_names

# de-select "year"
task$select(setdiff(task$feature_names, "year"))

task$feature_names

# Add new column "foo"
task$bind(data.frame(foo = 1:344))
task$head()
Description

This task specializes Task and TaskSupervised for classification problems. The target column is assumed to be a factor. The task_type is set to "classif".

Additional task properties include:

- "twoclass": The task is a binary classification problem.
- "multiclass": The task is a multiclass classification problem.

Predefined tasks are stored in the dictionary mlr_tasks. More example tasks can be found in this dictionary after loading mlr3data.

Super classes

mlr3::Task -> mlr3::TaskSupervised -> TaskClassif

Active bindings

class_names (character())
    Returns all class labels of the target column.

positive (character(1))
    Stores the positive class for binary classification tasks, and NA for multiclass tasks. To switch the positive class, assign a level to this field.

negative (character(1))
    Stores the negative class for binary classification tasks, and NA for multiclass tasks.

Methods

Public methods:

- TaskClassif$new()
- TaskClassif$data()
- TaskClassif$truth()
- TaskClassif$droplevels()
- TaskClassif$clone()

Method new(): Creates a new instance of this R6 class. The function as_task_classif() provides an alternative way to construct classification tasks.

Usage:
TaskClassif$new(id, backend, target, positive = NULL, extra_args = list())

Arguments:

id (character(1))
    Identifier for the new instance.
backend (DataBackend)
    Either a DataBackend, or any object which is convertible to a DataBackend with as_data_backend().
    E.g., a data.frame() will be converted to a DataBackendDataTable.

target (character(1))
    Name of the target column.

positive (character(1))
    Only for binary classification: Name of the positive class. The levels of the target columns
    are reordered accordingly, so that the first element of $class_names is the positive class, and
    the second element is the negative class.

extra_args (named list())
    Named list of constructor arguments, required for converting task types via convert_task().

Method data(): Calls $data from parent class Task and ensures that levels of the target column
are in the right order.

Usage:
TaskClassif$data(
    rows = NULL,
    cols = NULL,
    data_format = "data.table",
    ordered = TRUE
)

Arguments:
rows integer()
    Row indices.

cols character()
    Column names.

data_format (character(1))
    Desired data format, e.g. "data.table" or "Matrix".

ordered (logical(1))
    If TRUE (default), data is ordered according to the columns with column role "order".

Returns: Depending on the DataBackend, but usually a data.table::data.table().

Method truth(): True response for specified row_ids. Format depends on the task type.
Defaults to all rows with role "use".

Usage:
TaskClassif$truth(rows = NULL)

Arguments:
rows integer()
    Row indices.

Returns: factor().

Method droplevels(): Updates the cache of stored factor levels, removing all levels not present
in the current set of active rows. cols defaults to all columns with storage type "factor" or "ordered".
Also updates the task property "twoclass"/"multiclass".

Usage:
TaskClassif$droplevels(cols = NULL)

Arguments:
cols character()
  Column names.

Returns: Modified self.

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

Usage:
TaskClassif$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

See Also

Other Task: TaskRegr, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing, mlr_tasks_breast_cancer, mlr_tasks_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_penguins, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo, mlr_tasks

Examples

data("Sonar", package = "mlbench")
task = TaskClassif$new("sonar", backend = Sonar, target = "Class", positive = "M")

task$task_type
task$formula()
task$truth()
task$class_names
task$positive

# possible properties:
mlr_reflections$task_properties$task
Public fields

id (character(1))
Identifier of the object. Used in tables, plot and text output.

task_type (character(1))
Task type, e.g. \"classif\" or \"regr\". For a complete list of possible task types (depending on the loaded packages), see \texttt{mlr_reflections$task_types$type}.

param_set (paradox::ParamSet)
Set of hyperparameters.

packages (character(1))
Set of required packages. These packages are loaded, but not attached.

man (character(1))
String in the format \texttt{[pkg]\::[topic]} pointing to a manual page for this object. Defaults to \texttt{NA}, but can be set by child classes.

Methods

Public methods:

- \texttt{TaskGenerator$new()}  
- \texttt{TaskGenerator$format()}  
- \texttt{TaskGenerator$print()}  
- \texttt{TaskGenerator$generate()}  
- \texttt{TaskGenerator$clone()}

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

Usage:

\begin{verbatim}
TaskGenerator$new(
  id,  
  task_type,  
  packages = character(),  
  param_set = ParamSet$new(),  
  man = NA_character_,  
)
\end{verbatim}

Arguments:

id (character(1))
Identifier for the new instance.

task_type (character(1))
Type of task, e.g. \"regr\" or \"classif\". Must be an element of \texttt{mlr_reflections$task_types$type}.

packages (character())
Set of required packages. A warning is signaled by the constructor if at least one of the packages is not installed, but loaded (not attached) later on-demand via \texttt{requireNamespace()}.  

param_set (paradox::ParamSet)
Set of hyperparameters.

man (character(1))
String in the format \texttt{[pkg]\::[topic]} pointing to a manual page for this object. The referenced help package can be opened via method $help().
Method format(): Helper for print outputs.
Usage:
TaskGenerator$format()

Method print(): Printer.
Usage:
TaskGenerator$print(...)
Arguments:
... (ignored).

Method generate(): Creates a task of type task_type with n observations, possibly using additional settings stored in param_set.
Usage:
TaskGenerator$generate(n)
Arguments:
  n (integer(1))  
  Number of rows to generate.
Returns: Task.

Method clone(): The objects of this class are cloneable with this method.
Usage:
TaskGenerator$clone(deep = FALSE)
Arguments:
  deep  Whether to make a deep clone.

See Also
Other TaskGenerator: mlr_task_generators_2dnormals, mlr_task_generators_cassini, mlr_task_generators_circle, mlr_task_generators_friedman1, mlr_task_generators_moons, mlr_task_generators_simplex, mlr_task_generators_smiley, mlr_task_generators_spirals, mlr_task_generators_xor, mlr_task_generators

TaskRegr  Regression Task

Description
This task specializes Task and TaskSupervised for regression problems. The target column is assumed to be numeric. The task_type is set to "regr".

Predefined tasks are stored in the dictionary mlr_tasks. More example tasks can be found in this dictionary after loading mlr3data.
Super classes

mlr3::Task -> mlr3::TaskSupervised -> TaskRegr

Methods

Public methods:

- TaskRegr$new()
- TaskRegr$truth()
- TaskRegr$clone()

Method new(): Creates a new instance of this R6 class. The function as_task_regr() provides an alternative way to construct regression tasks.

Usage:
TaskRegr$new(id, backend, target, extra_args = list())

Arguments:
- id (character(1))
  Identifier for the new instance.
- backend (DataBackend)
  Either a DataBackend, or any object which is convertible to a DataBackend with as_data_backend().
  E.g., a data.frame() will be converted to a DataBackendDataTable.
- target (character(1))
  Name of the target column.
- extra_args (named list())
  Named list of constructor arguments, required for converting task types via convert_task().

Method truth(): True response for specified row_ids. Format depends on the task type. Defaults to all rows with role "use".

Usage:
TaskRegr$truth(rows = NULL)

Arguments:
- rows integer()
  Row indices.

Returns: numeric().

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

Usage:
TaskRegr$clone(deep = FALSE)

Arguments:
- deep Whether to make a deep clone.

See Also

Other Task: TaskClassif, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing, mlr_tasks_breast_cancer, mlr_tasks_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_penguins, mlr_tasks_pima, mlr_tasks_iris, mlr_tasks_zoo, mlr_tasks_wine, mlr_tasks_spam, mlr_tasks_zoo, mlr_tasks
Examples

```r
  task = TaskRegr$new("penguins", backend = palmerpenguins::penguins, target = "bill_length_mm")
  task$task_type
  task$formula()
  task$truth()

  # possible properties:
  mlr_reflections$task_properties$regr
```
## Index

* **DataBackend**
  - as_data_backend.Matrix, 8
  - DataBackend, 31
  - DataBackendDataTable, 32
  - DataBackendMatrix, 35

* **Dictionary**
  - mlr_learners, 56
  - mlr_measures, 66
  - mlr_resamplings, 126
  - mlr_task_generators, 155
  - mlr_tasks, 142

* **Learner**
  - Learner, 38
  - LearnerClassif, 44
  - LearnerRegr, 46
  - mlr_learners, 56
  - mlr_learners_classif.debug, 57
  - mlr_learners_classif.featureless, 59
  - mlr_learners_classif.rpart, 61
  - mlr_learners_regr.featureless, 63
  - mlr_learners_regr.rpart, 64

* **Measure**
  - Measure, 48
  - MeasureClassif, 52
  - MeasureRegr, 54
  - mlr_measures, 66
  - mlr_measures_classif.costs, 73
  - mlr_measures_debug, 101
  - mlr_measures_elapsed_time, 102
  - mlr_measures_oob_error, 104
  - mlr_measures_selected_features, 125

* **Prediction**
  - Prediction, 169
  - PredictionClassif, 171
  - PredictionRegr, 175

* **Resampling**
  - mlr_resamplings, 126
  - mlr_resamplings_bootstrap, 127
  - mlr_resamplings_custom, 129
  - mlr_resamplings_cv, 131
  - mlr_resamplings_holdout, 132
  - mlr_resamplings_insample, 134
  - mlr_resamplings_loo, 135
  - mlr_resamplings_repeated_cv, 137
  - mlr_resamplings_subsampling, 139

* **TaskGenerator**
  - mlr_task_generators, 155
  - mlr_task_generators_2dnormals, 156
  - mlr_task_generators_cassini, 157
  - mlr_task_generators_circle, 158
  - mlr_task_generators_friedman1, 160
  - mlr_task_generators_moons, 161
  - mlr_task_generators_simplex, 162
  - mlr_task_generators_smiley, 164
  - mlr_task_generators_spirals, 165
  - mlr_task_generators_xor, 167
  - TaskGenerator, 199

* **Task**
  - mlr_tasks, 142
  - mlr_tasks_boston_housing, 144
  - mlr_tasks_breast_cancer, 145
  - mlr_tasks_german_credit, 146
  - mlr_tasks_iris, 147
  - mlr_tasks_mtcars, 148
  - mlr_tasks_penguins, 149
  - mlr_tasks_pima, 150
  - mlr_tasks_sonar, 151
  - mlr_tasks_spam, 152
  - mlr_tasks_wine, 153
  - mlr_tasks_zoo, 154
  - Task, 187
  - TaskClassif, 197
  - TaskRegr, 201

* **binary classification measures**
  - mlr_measures_classif.auc, 68
mlr_measures_classif.bbrier, 70
mlr_measures_classif.dor, 75
mlr_measures_classif.fbeta, 76
mlr_measures_classif.fdr, 77
mlr_measures_classif.fn, 78
mlr_measures_classif.fnr, 80
mlr_measures_classif.fp, 82
mlr_measures_classif.fpr, 83
mlr_measures_classif.mcc, 87
mlr_measures_classif.npv, 88
mlr_measures_classif.ppv, 89
mlr_measures_classif.prauc, 90
mlr_measures_classif.precision, 91
mlr_measures_classif.recall, 92
mlr_measures_classif.sensitivity, 94
mlr_measures_classif.specificity, 95
mlr_measures_classif.tn, 96
mlr_measures_classif.tnr, 97
mlr_measures_classif.tp, 98
mlr_measures_classif.tpr, 100

* classification measures
mlr_measures_classif.acc, 67
mlr_measures_classif.auc, 68
mlr_measures_classif.bacc, 69
mlr_measures_classif.bbrier, 70
mlr_measures_classif.ce, 72
mlr_measures_classif.costs, 73
mlr_measures_classif.dor, 75
mlr_measures_classif.fbeta, 76
mlr_measures_classif.fdr, 77
mlr_measures_classif.fn, 78
mlr_measures_classif.fnr, 80
mlr_measures_classif.fomr, 98
mlr_measures_classif.fp, 82
mlr_measures_classif.fpr, 83
mlr_measures_classif.logloss, 84
mlr_measures_classif.mbrier, 85
mlr_measures_classif.mcc, 87
mlr_measures_classif.npv, 88
mlr_measures_classif.ppv, 89
mlr_measures_classif.prauc, 90
mlr_measures_classif.precision, 91
mlr_measures_classif.recall, 92
mlr_measures_classif.sensitivity, 94
mlr_measures_classif.tn, 96
mlr_measures_classif.tnr, 97
mlr_measures_classif.tp, 98
mlr_measures_classif.tpr, 100

mlr_measures_classif specificity, 95
mlr_measures_classif.tn, 96
mlr_measures_classif.tnr, 97
mlr_measures_classif.tp, 98
mlr_measures_classif.tpr, 100

* datasets
mlr_learners, 56
mlr_measures, 66
mlr_resamplings, 126
mlr_task_generators, 155
mlr_tasks, 142

* multiclass classification measures
mlr_measures_classif.acc, 67
mlr_measures_classif.bacc, 69
mlr_measures_classif.ce, 72
mlr_measures_classif.costs, 73
mlr_measures_classif.logloss, 84
mlr_measures_classif.mbrier, 85

* regression measures
mlr_measures_regr.bias, 105
mlr_measures_regr.ktau, 106
mlr_measures_regr.mae, 107
mlr_measures_regr.mape, 108
mlr_measures_regr.maxae, 109
mlr_measures_regr.medae, 110
mlr_measures_regr.medianae, 111
mlr_measures_regr.medianae, 112
mlr_measures_regr.maxae, 113
mlr_measures_regr.pbias, 114
mlr_measures_regr.rae, 115
mlr_measures_regr.rmse, 116
mlr_measures_regr.rmsle, 117
mlr_measures_regr.rsq, 120
mlr_measures_regr.rmse, 119
mlr_measures_regr.rsq, 120
mlr_measures_regr.sae, 121
mlr_measures_regr.smape, 122
mlr_measures_regr.srho, 123
mlr_measures_regr.sse, 124

as_benchmark_result, 8
as_benchmark_result(), 16
as_data_backend
(as_data_backend.Matrix), 8
as_data_backend(), 31
as_data_backend.Matrix, 8, 32, 34, 37
as_learner, 9
as_learners (as_learner), 9
as_measure, 10
as_measures (as_measure), 10
as_prediction, 11
as_prediction(), 174
as_prediction_classif, 12
as_prediction_data, 13
as_prediction_data(), 174
as_prediction_regr, 14
as_predictions (as_prediction), 11
as_resample_result, 15
as_resample_result(), 180
as_resampling, 16
as_res samplings (as_resampling), 16
as_result_data, 16
as_result_data(), 25, 180
as_task, 18
as_task_classif, 18
as_task_classif(), 197
as_task_regr, 20
as_task_regr(), 202
as_tasks (as_task), 18

bbrier(), 86
benchmark, 21
benchmark(), 12, 24, 26, 40, 43, 49, 51, 54, 56, 182
benchmark_grid, 29
benchmark_grid(), 21
BenchmarkResult, 8, 16, 22, 24, 25, 26, 40, 43, 49, 179, 180
bootstrap, 182
boston_housing, 188

c(), 26
c.PredictionDataClassif (PredictionData), 174
c.PredictionDataRegr (PredictionData), 174
check_prediction_data (PredictionData), 174
classif.auc, 48
classif.ce, 54
classif.rpart, 38
convert_task, 30
convert_task(), 18, 20, 189, 192, 198, 202
cv, 182
data.frame(), 9, 18, 20, 21, 36, 168
data.table(), 33, 180
data.table::as.data.table(), 9
data.table::copy(), 33
data.table::data.table(), 9, 21, 24–27, 30–34, 36, 40, 56, 67, 127, 143, 155, 169, 179, 181, 188–193, 198
DataBackend, 8, 9, 18, 20, 22, 31, 32, 34, 35, 37, 177, 188, 189, 192–194, 198, 202
DataBackendDataTable, 8, 9, 31, 32, 37, 188, 192, 198, 202
DataBackendMatrix, 8, 9, 31, 32, 34, 35
datasets::iris, 147
datasets::mtcars, 148
default_measures, 37
default_measures(), 181
distr6::VectorDistribution, 46, 175, 176
expand.grid(), 29
formula(), 192, 193
future::multicore, 187
future::plan, 187
future::plan(), 22, 177
intersect(), 189, 190
iris data set, 149
is_missing_prediction_data (PredictionData), 174
Learner, 9, 10, 12, 17, 21, 24, 26, 28, 29, 38, 40, 42, 44–49, 51–57, 59, 61–64, 66, 104, 125, 141, 168, 170, 171, 174, 176, 177, 179, 181, 187
LearnerClassif, 38, 41, 44, 48, 57, 59, 61, 62, 64, 66, 171
LearnerClassifDebug (mlr_learners_classif.debug), 57
mlr_measures::medae(), 110
mlr_measures::medse(), 111
mlr_measures::mse(), 112
mlr_measures::msle(), 113
mlr_measures::npv(), 88
mlr_measures::pbias(), 114
mlr_measures::ppv(), 89
mlr_measures::mrauc(), 91
mlr_measures::precision(), 92
mlr_measures::rae(), 115
mlr_measures::recall(), 93
mlr_measures::rmse(), 116
mlr_measures::rmsle(), 117
mlr_measures::rse(), 118
mlr_measures::rse(), 119
mlr_measures::rsq(), 120
mlr_measures::sae(), 121
mlr_measures::sensitivity(), 94
mlr_measures::smape(), 122
mlr_measures::specificity(), 95
mlr_measures::srho(), 123
mlr_measures::sse(), 124
mlr_measures::tn(), 97
mlr_measures::tnr(), 98
mlr_measures::tp(), 99
mlr_measures::tpn(), 100
mlrmisc::Dictionary, 56, 66, 67, 126, 127, 141, 143, 155
mlrmisc::dictionary_sugar_get(), 141, 142
mlrmisc::encapsulate(), 40, 41
mlrmisc::insert_named(), 39
mlrmisc::unnest(), 27
mlr_learners, 38, 44, 46, 48, 56, 57, 59, 61–64, 66, 67, 127, 141, 143, 155
mlr_learners_classif.debug, 44, 46, 48, 57, 59, 61, 62, 64, 66
mlr_learners_classif.featureless, 44, 46, 48, 57, 59, 61, 62, 64, 66
mlr_learners_classif.rpart, 44, 46, 48, 57, 59, 61, 61, 64, 66
mlr_learners_regr.featureless, 44, 46, 48, 57, 59, 61, 61, 62, 6, 64, 66
mlr_learners_regr.rpart, 44, 46, 48, 57, 59, 61, 62, 64, 64
mlr_measures, 48, 52, 54, 56, 57, 58, 67–127, 141, 143, 155
mlr_measures_classif.auc, 68, 68, 70–72, 74, 76–101
mlr_measures_classif.brier, 68–70, 70, 72, 74, 76–101
mlr_measures_classif.ce, 68–71, 72, 74, 76–80, 82–88, 90–94, 96–100
mlr_measures_classif.costs, 52, 54, 56, 67–73, 73, 76–80, 82–88, 90–94, 96–100, 102, 103, 105, 126
mlr_measures_classif.dor, 68–72, 74, 75, 77–101
mlr_measures_classif.fbeta, 68–72, 74, 76, 76, 78–101
mlr_measures_classif.fdr, 68–72, 74, 76, 77, 79–101
mlr_measures_classif.fn, 68–72, 74, 76–78, 78, 80–101
mlr_measures_classif.fnr, 68–72, 74, 76–79, 80, 82–101
mlr_measures_classif.fp, 68–72, 74, 76–81, 81, 83–101
mlr_measures_classif.fpr, 68–72, 74, 76–83, 83, 85–101
mlr_measures_classif.fomr, 68–72, 74, 76–91, 91, 92–101
mlr_measures_classif.fomr, 68–72, 74, 76–89, 89, 90–101
mlr_measures_classif.fomr, 68–72, 74, 76–89, 89, 90–101
mlr_measures_classif.fomr, 68–72, 74, 76–89, 89, 90–101
mlr_measures_classif.fomr, 68–72, 74, 76–89, 89, 90–101
mlr_measures_classif.fomr, 68–72, 74, 76–89, 89, 90–101
mlr_measures_classif.fomr, 68–72, 74, 76–89, 89, 90–101
mlr_measures_classif.fomr, 68–72, 74, 76–89, 89, 90–101
mlr_measures_classif.logloss, 68–74, 76–80, 82–84, 84, 86, 87, 89–94, 96–100
mlr_measures_classif.mbrier, 68–71, 73, 74, 76–80, 82–85, 85, 87, 89–94, 96–100
mlr_measures_classif.mcc, 68–74, 76–86, 87, 89–101
mlr_measures_classif.npv, 68–74, 76–88, 88, 90–101
mlr_measures_classifppv, 68–74, 76–89, 89, 91–101
mlr_measures_classif.pauc, 68–74, 76–90, 90–101
mlr_measures_classif.precision, 68–74, 76–91, 91, 93–101
mlr_measures_classif.ppv, 68–74, 76–92, 92, 94–101
mlr_measures_classif.sensitivity,
INDEX

68–74, 76–79, 81–93, 94, 96–101
mlr_measures_classif.specificity, 68–74, 76–79, 81–95, 95, 97–101
mlr_measures_classif.tnr, 68–74, 76–79, 81–97, 97, 99–101
mlr_measures_classif.tp, 68–74, 76–79, 81–98, 98, 100, 101
mlr_measures_classif.tpr, 68–74, 76–79, 81–99, 100
mlr_measures_debug, 52, 54, 56, 67, 74, 101, 103, 105, 126
mlr_measures_elapsed_time, 52, 54, 56, 67, 74, 102, 102, 105, 126
mlr_measures_oob_error, 52, 54, 56, 67, 74, 102, 103, 104, 126
mlr_measures_regr.bias, 105, 107–125
mlr_measures_regr.ktau, 106, 106, 108–125
mlr_measures_regr.mae, 106, 107, 107, 109–125
mlr_measures_regr.maxae, 106–109, 109, 111–125
mlr_measures_regr.medae, 106–110, 110, 112–125
mlr_measures_regr.medse, 106–111, 111, 113–125
mlr_measures_regr.mse, 106–112, 112, 114–125
mlr_measures_regr.msle, 106–113, 113, 115–125
mlr_measures_regr.pbias, 106–114, 114, 116–125
mlr_measures_regr.rae, 106–115, 115, 117–125
mlr_measures_regr.rmse, 106–116, 116, 118–125
mlr_measures_regr.rmsle, 106–117, 117, 119–125
mlr_measures_regr.rrse, 106–118, 118, 120–125
mlr_measures_regr.rse, 106–119, 119, 121–125
mlr_measures_regr.rsq, 106–120, 120, 122–125
mlr_measures_regr.sae, 106–121, 121, 123–125
mlr_measures_regr.srho, 106–122, 122, 124, 125
mlr_measures_regr.smape, 106–123, 123, 125
mlr_measures_regr.sse, 106–124, 124
mlr_measures_selected_features, 52, 54, 56, 67, 74, 102, 103, 105, 125
mlr_measures_time_both
(mlr_measures_elapsed_time), 102
mlr_measures_time_predict
(mlr_measures_elapsed_time), 102
mlr_measures_time_train
(mlr_measures_elapsed_time), 102
mlr_reflections, 168
mlr_reflections$default_measures, 11, 37
mlr_reflections$learner_predict_types, 40, 42, 45, 47, 51, 54, 55
mlr_reflections$learner_properties, 40, 42, 45, 47
mlr_reflections$measure_properties, 51, 53, 55
mlr_reflections$task_feature_types, 40, 42, 45, 47, 188
mlr_reflections$task_properties, 189
mlr_reflections$task_types, 30
mlr_reflections$task_types$type, 39, 42, 49, 50, 188, 191, 200
mlr_resamplings, 57, 67, 126, 127–137, 139–141, 143, 155, 182, 186
mlr_resamplings$bootstrap, 127, 127, 130, 132, 133, 135, 136, 139, 140, 186
mlr_resamplings$custom, 127, 128, 129, 132, 133, 135, 136, 139, 140, 186
mlr_resamplings$cv, 127, 128, 130, 131, 133, 135, 136, 139, 140, 186
mlr_resamplings$holdout, 127, 128, 130, 132, 132, 135, 136, 139, 140, 186
mlr_resamplings$in.sample, 127, 128, 130, 132, 133, 134, 136, 139, 140, 186
mlr_resamplings$loo, 127, 128, 130, 132, 133, 135, 139, 140, 186
mlr_resamplings$repeated_cv, 127, 128,
mlr_tasks, 127, 128, 130, 132, 133, 135, 136, 139, 139, 186
mlr_resamplings_subsampling, 127, 128, 130, 132, 133, 135, 136, 139, 139, 186
mlr_sugar, 141
mlr_task_generators, 57, 67, 127, 141, 143, 155, 156–168, 199, 201
mlr_task_generators_2dnormals, 155, 156, 158, 159, 161–163, 165, 166, 168, 201
mlr_task_generators_cassini, 155, 157, 157, 159, 161–163, 165, 166, 168, 201
mlr_task_generators_circle, 155, 157, 158, 158, 161–163, 165, 166, 168, 201
mlr_task_generators_friedman1, 155, 157–159, 160, 162, 163, 165, 166, 168, 201
mlr_task_generators_moons, 155, 157–159, 161, 161, 163, 165, 166, 168, 201
mlr_task_generators_simplex, 155, 157–159, 161, 162, 162, 165, 166, 168, 201
mlr_task_generators_smiley, 155, 157–159, 161–163, 164, 166, 168, 201
mlr_task_generators_spirals, 155, 157–159, 161–163, 165, 165, 168, 201
mlr_task_generators_xor, 155, 157–159, 161–163, 165, 166, 167, 201
mlr_tasks, 57, 67, 127, 141, 142, 144, 145, 147, 148, 150, 151, 153–155, 188, 196, 197, 199, 201, 202
mlr_tasks_boston_housing, 143, 144, 145, 147, 148, 150, 151, 153–155, 196, 199, 202
mlr_tasks_breast_cancer, 143, 144, 145, 147, 148, 150, 151, 153–155, 196, 199, 202
mlr_tasks_german_credit, 143–145, 146, 148, 150, 151, 153–155, 196, 199, 202
mlr_tasks_iris, 143–145, 147, 147, 148, 150, 151, 153–155, 196, 199, 202
mlr_tasks_mtcars, 143–145, 147, 148, 148, 150, 151, 153–155, 196, 199, 202
mlr_tasks_penguins, 143–145, 147, 148, 149, 150, 151, 153–155, 196, 199, 202
mlr_tasks_pima, 143–145, 147, 148, 150, 150, 151, 153–155, 196, 199, 202
mlr_tasks_sonar, 143–145, 147, 148, 150, 151, 153–155, 196, 199, 202
mlr_tasks_spam, 143–145, 147, 148, 150, 151, 152, 154, 155, 196, 199, 202
mlr_tasks_wine, 143–145, 147, 148, 150, 151, 153, 155, 196, 199, 202
msr (mlr_sugar), 141
msrs (mlr_sugar), 141
msrs(), 66, 67
palmerpenguins::penguins, 149
paradox::ParamSet, 38, 39, 41, 42, 45, 47, 142, 183, 184, 200
parallelly::availableCores(), 187
ParamSet, 39
penguins, 188
plot(), 156, 158, 159, 162–167
precision(), 76
predict.Learner, 168
Prediction, 11, 12, 17, 24, 26, 40, 43, 44, 46, 49–53, 55, 101, 168, 169, 169, 171, 173, 174, 176, 180, 181
PredictionClassif, 12, 13, 44, 169, 171, 171, 176
PredictionData, 13, 14, 174, 175
PredictionRegr, 14, 46, 169, 171, 173, 175
progressr::with_progress(), 22, 177
R6::R6Class, 56, 66, 126, 142–155
recall(), 76
regr.mse, 56
regr.rpart, 38
resample(). 12, 26, 40, 43, 49, 51, 54, 56, 178, 179, 182
ResampleResult. 12, 15–17, 24–28, 40, 43, 49–53, 177, 178, 179, 180, 182
Resampling. 16, 17, 21, 25, 26, 28, 29, 51, 53, 55, 126–137, 139–141, 176, 177, 179, 181, 182, 183, 186, 190
ResamplingBootstrap. 182
ResamplingBootstrap
(mlr_resamplings_bootstrap), 127
ResamplingCustom
(mlr_resamplings_custom), 129
ResamplingCV. 182, 184
ResamplingCV (mlr_resamplings_cv), 131
ResamplingHoldout, 184
ResamplingHoldout
(mlr_resamplings_holdout), 132
ResamplingInsample
(mlr_resamplings_insampling), 134
ResamplingLOO (mlr_resamplings_loo), 135
ResamplingRepeatedCV
(mlr_resamplings_repeated_cv), 137
Resamplings. 128, 130, 132, 133, 135, 136, 139, 140, 186
ResamplingSubsampling
(mlr_resamplings_subsampling), 139
ResultData, 16, 179, 180
rpart::rpart(), 61
rse(), 120
rsmp (mlr_sugar), 141
rsmp(), 126, 127, 129, 131, 132, 134, 136, 137, 139
rsmps (mlr_sugar), 141
rsmps(), 126, 127
sd(), 63
set_threads, 186
setdiff(), 189, 190
stats::cor(), 106, 123
stats::predict(), 168
Task. 14, 17, 18, 21, 22, 24, 26–31, 43, 49, 51–56, 73, 125, 130, 141, 143–145,
TaskClassif. 18–20, 143–155, 173, 187, 191, 196, 197, 202
TaskGenerator. 141, 155–168, 199
TaskGenerator2DNormals
(mlr_task_generators_2dnormals), 156
TaskGeneratorCassini
(mlr_task_generators_cassini), 157
TaskGeneratorCircle
(mlr_task_generators_circle), 158
TaskGeneratorFriedman1
(mlr_task_generators_friedman1), 160
TaskGeneratorMoons
(mlr_task_generators_moons), 161
TaskGenerators. 157–159, 161–163, 165, 166, 168
TaskGeneratorSimplex
(mlr_task_generators_simplex), 162
TaskGeneratorSmiley
(mlr_task_generators_smiley), 164
TaskGeneratorSpirals
(mlr_task_generators_spirals), 165
TaskGeneratorXor
(mlr_task_generators_xor), 167
TaskRegr. 18, 20, 21, 143–145, 147, 148, 150, 151, 153–155, 176, 187, 191, 196, 199, 201
Tasks. 144, 145, 147, 148, 150, 151, 153–155
TaskSupervised. 143–145, 147, 148, 150, 151, 153–155, 187, 196, 197, 199, 201, 202
tgen (mlr_sugar), 141
tgen(), 155–157, 159–161, 163–165, 167
tgens (mlr_sugar), 141
tgens(), 155
time_train, 48
tsk (mlr_sugar), 141
 tsk(), 143
 tsks (mlr_sugar), 141
 tsks(), 143
 union(), 189, 190

VectorDistribution, 176

xor, 199