Package ‘mlr3’

April 17, 2020

Title Machine Learning in R - Next Generation
Version 0.2.0
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 (>= 1.12.8), digest, lgr (>= 0.3.4), mlbench, mlr3measures (>= 0.1.3), mlr3misc (>= 0.2.0), paradox (>= 0.2.0), uuid
Suggests Matrix, bibtex, callr, datasets, evaluate, future (>= 1.16.0), future.apply (>= 1.4.0), future.callr, mlr3data, progressr, rpart, testthat
RdMacros mlr3misc
Encoding UTF-8
LazyData true
NeedsCompilation no
RoxygenNote 7.1.0
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'
'LearnerRegr.R' 'LearnerRegrFeatureless.R' 'LearnerRegrRpart.R'
R topics documented:

`'Measure.R' 'MeasureClassif.R' 'mlr_measures.R'
'MeasureClassifCosts.R' 'MeasureDebug.R' 'MeasureElapsedTime.R'
'MeasureOOBEError.R' 'MeasureRegr.R' 'MeasureSelectedFeatures.R'
'MeasureSimple.R' 'Prediction.R' 'PredictionClassif.R'
'PredictionRegr.R' 'ResampleResult.R' 'Resampling.R'
'mlr_resamplings.R' 'ResamplingBootstrap.R' 'ResamplingCV.R'
'ResamplingCustom.R' 'ResamplingHoldout.R'
'ResamplingInsample.R' 'ResamplingRepeatedCV.R'
'ResamplingSubsampling.R' 'Task.R' 'TaskSupervised.R'
'TaskClassif.R' 'mlr_tasks.R' 'TaskClassif_german_credit.R'
'TaskClassif_iris.R' 'TaskClassif_pima.R' 'TaskClassif_sonar.R'
'TaskClassif_spam.R' 'TaskClassif_wine.R' 'TaskClassif_zoo.R'
'TaskGenerator.R' 'mlr_task_generators.R'
'TaskGenerator2DNormals.R' 'TaskGeneratorFriedman1.R'
'TaskGeneratorSmiley.R' 'TaskGeneratorXor.R' 'TaskRegr.R'
'TaskRegr_boston_housing.R' 'TaskRegr_mtcars.R'
'TaskUnsupervised.R' 'Task_cbind.R' 'Task_rbind.R'
'as_data_backend.R' 'assertions.R' 'auto_convert.R'
'benchmark.R' 'benchmark_grid.R' 'default_measures.R'
'deprecated.R' 'fix_factor_levels.R' 'helper.R'
'mlr_coercions.R' 'mlr_sugar.R' 'predict.R' 'reexports.R'
'resample.R' 'worker.R' 'zzz.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]

**Maintainer**  Michel Lang <michellang@gmail.com>

**Repository**  CRAN

**Date/Publication**  2020-04-17 13:40:06 UTC

---

R topics documented:

  - `mlr3-package` .................................................. 5
  - `as_benchmark_result` ........................................ 6
  - `as_data_backend.data.frame` ................................ 7
  - `benchmark` ...................................................... 8
  - `BenchmarkResult` .............................................. 10
  - `benchmark_grid` ............................................... 15
  - `DataBackend` ................................................... 16
  - `DataBackendDataTable` ...................................... 18
  - `DataBackendMatrix` .......................................... 20
  - `default_measures` ............................................ 23
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learner</td>
<td>24</td>
</tr>
<tr>
<td>LearnerClassif</td>
<td>29</td>
</tr>
<tr>
<td>LearnerRegr</td>
<td>32</td>
</tr>
<tr>
<td>Measure</td>
<td>34</td>
</tr>
<tr>
<td>MeasureClassif</td>
<td>38</td>
</tr>
<tr>
<td>MeasureRegr</td>
<td>39</td>
</tr>
<tr>
<td>mlr_coercions</td>
<td>41</td>
</tr>
<tr>
<td>mlr_learners</td>
<td>43</td>
</tr>
<tr>
<td>mlr_learners_classif.debug</td>
<td>44</td>
</tr>
<tr>
<td>mlr_learners_classif.featureless</td>
<td>46</td>
</tr>
<tr>
<td>mlr_learners_classif.rpart</td>
<td>47</td>
</tr>
<tr>
<td>mlr_learners_regr.featureless</td>
<td>49</td>
</tr>
<tr>
<td>mlr_learners_regr.rpart</td>
<td>50</td>
</tr>
<tr>
<td>mlr_measures</td>
<td>51</td>
</tr>
<tr>
<td>mlr_measures_classif.acc</td>
<td>52</td>
</tr>
<tr>
<td>mlr_measures_classif.auc</td>
<td>53</td>
</tr>
<tr>
<td>mlr_measures_classif.bacc</td>
<td>54</td>
</tr>
<tr>
<td>mlr_measures_classif.bbrier</td>
<td>55</td>
</tr>
<tr>
<td>mlr_measures_classif.ce</td>
<td>57</td>
</tr>
<tr>
<td>mlr_measures_classif.costs</td>
<td>58</td>
</tr>
<tr>
<td>mlr_measures_classif.dor</td>
<td>60</td>
</tr>
<tr>
<td>mlr_measures_classif.fbeta</td>
<td>61</td>
</tr>
<tr>
<td>mlr_measures_classif.fdr</td>
<td>62</td>
</tr>
<tr>
<td>mlr_measures_classif.fn</td>
<td>63</td>
</tr>
<tr>
<td>mlr_measures_classif.fnr</td>
<td>64</td>
</tr>
<tr>
<td>mlr_measures_classif.fomr</td>
<td>66</td>
</tr>
<tr>
<td>mlr_measures_classif.fp</td>
<td>67</td>
</tr>
<tr>
<td>mlr_measures_classif.fpr</td>
<td>68</td>
</tr>
<tr>
<td>mlr_measures_classif.logloss</td>
<td>69</td>
</tr>
<tr>
<td>mlr_measures_classif.mbrier</td>
<td>70</td>
</tr>
<tr>
<td>mlr_measures_classif.mcc</td>
<td>71</td>
</tr>
<tr>
<td>mlr_measures_classif.npv</td>
<td>72</td>
</tr>
<tr>
<td>mlr_measures_classif.ppv</td>
<td>74</td>
</tr>
<tr>
<td>mlr_measures_classif.precision</td>
<td>75</td>
</tr>
<tr>
<td>mlr_measures_classif.recall</td>
<td>76</td>
</tr>
<tr>
<td>mlr_measures_classif.sensitivity</td>
<td>77</td>
</tr>
<tr>
<td>mlr_measures_classif.specificity</td>
<td>78</td>
</tr>
<tr>
<td>mlr_measures_classif.tn</td>
<td>80</td>
</tr>
<tr>
<td>mlr_measures_classif.tnr</td>
<td>81</td>
</tr>
<tr>
<td>mlr_measures_classif.tp</td>
<td>82</td>
</tr>
<tr>
<td>mlr_measures_classif.tpr</td>
<td>83</td>
</tr>
<tr>
<td>mlr_measures_debug</td>
<td>84</td>
</tr>
<tr>
<td>mlr_measures_elapsed_time</td>
<td>86</td>
</tr>
<tr>
<td>mlr_measures_oob_error</td>
<td>88</td>
</tr>
<tr>
<td>mlr_measures_regr.bias</td>
<td>89</td>
</tr>
<tr>
<td>mlr_measures_regr.ktau</td>
<td>90</td>
</tr>
<tr>
<td>mlr_measures_regr.mae</td>
<td>91</td>
</tr>
</tbody>
</table>
R topics documented:

mlr_measures_regr.maxae .................................................. 92
mlr_measures_regr.medae .................................................. 93
mlr_measures_regr.medse .................................................. 94
mlr_measures_regr.mse .................................................... 95
mlr_measures_regr.msle ................................................... 96
mlr_measures_regr.pbias ................................................... 97
mlr_measures_regr.rae ..................................................... 98
mlr_measures_regr.rmse .................................................... 99
mlr_measures_regr.rmsle .................................................. 100
mlr_measures_regr.rse ..................................................... 101
mlr_measures_regr.rse ..................................................... 102
mlr_measures_regr.rsq .................................................... 103
mlr_measures_regr.sae ..................................................... 104
mlr_measures_regr.smape .................................................. 105
mlr_measures_regr.srho ................................................... 106
mlr_measures_regr.sse ..................................................... 107
mlr_measures_selected_features ........................................ 108
mlr_resamplings ........................................................... 109
mlr_resamplings_bootstrap .............................................. 110
mlr_resamplings_custom .................................................. 112
mlr_resamplings_cv ......................................................... 114
mlr_resamplings_holdout .................................................. 115
mlr_resamplings_insample ............................................... 117
mlr_resamplings_repeated_cv ......................................... 118
mlr_resamplings_subsampling ......................................... 121
mlr_sugar ................................................................. 122
mlr_tasks ................................................................. 124
mlr_tasks_boston_housing ............................................... 125
mlr_tasks_german_credit ............................................... 126
mlr_tasks_iris ............................................................ 127
mlr_tasks_mtcars ......................................................... 127
mlr_tasks_pima ............................................................ 128
mlr_tasks_sonar ........................................................... 128
mlr_tasks_spam ............................................................. 129
mlr_tasks_wine ............................................................. 130
mlr_tasks_zoo ............................................................... 131
mlr_task_generators ....................................................... 131
mlr_task_generators_2dnormals ....................................... 132
mlr_task_generators_friedman1 ........................................ 133
mlr_task_generators_smiley ............................................. 134
mlr_task_generators_xor .................................................. 135
predict.Learner ............................................................. 136
Prediction ................................................................. 137
PredictionClassif .......................................................... 139
PredictionRegr ............................................................. 142
resample ................................................................. 143
ResampleResult ............................................................ 145
Resampling ............................................................... 149
mlr3-package

Task ................................................................. 153
TaskClassif ...................................................... 161
TaskGenerator .................................................... 163
TaskRegr .......................................................... 165

Index 167

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.

Additional resources
• Book on mlr3: https://mlr3book.mlr-org.com
• Use cases and examples: https://mlr3gallery.mlr-org.com
• More classification and regression learners: mlr3learners
• Preprocessing and machine learning pipelines: mlr3pipelines
• Tuning of hyperparameters: mlr3tuning
• Visualizations for many mlr3 objects: mlr3viz
• Survival analysis and probabilistic regression: mlr3proba
• Feature selection filters: mlr3filters
• Interface to real (out-of-memory) data bases: mlr3db
• Performance measures as plain functions: mlr3measures
• Parallelization framework: future
• Progress bars: progressr

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]

References


See Also

Useful links:

- https://mlr3.mlr-org.com
- https://github.com/mlr-org/mlr3

---

**as_benchmark_result**

Convert to BenchmarkResult

**Description**

Simple S3 method to convert objects to a `BenchmarkResult`.

**Usage**

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

## S3 method for class 'ResampleResult'

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

**Arguments**

- `x` (any)
  Object to dispatch on, e.g. a `ResampleResult`.

- `...` (any)
  Currently not used.

**Value**

`BenchmarkResult`. 
as_data_backend.data.frame

Create a Data Backend

Description

Wraps a DataBackend around data.

Usage

```r
## S3 method for class 'data.frame'
as_data_backend(data, primary_key = NULL, keep_rownames = FALSE, ...)
```

```r
## S3 method for class 'Matrix'
as_data_backend(data, primary_key = NULL, dense = NULL, ...)
```

Arguments

- `data` any
  Data to create a DataBackend from. For a data.frame() (this includes tibble() from tibble and data.table::data.table()), a DataBackendDataTable is created. See methods("as_data_backend") for possible input formats.
  Package mlr3db extends this function with a method for lazy table objects implemented in dbplyr. This allows to interface many different data base systems such as SQL servers.

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

- `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 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 data.table::as.data.table().

- `...` (any)
  Additional arguments passed to the respective DataBackend method.

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

Value

DataBackend.

See Also

Other DataBackend: DataBackendDataTable, DataBackendMatrix, DataBackend
Examples

# create a new backend using the iris data:
as_data_backend(iris)

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)

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))
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.

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":

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

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

```r
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("iris", "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 iris with a 3-fold CV
# - fit classif.rpart on sonar using a holdout
tasks = list(tsk("iris"), tsk("sonar"))
learners = list(lrn("classif.featureless"), lrn("classif.rpart"))
resamplings = list(rsmp("cv", folds = 3), rsmp("holdout"))

design = data.table::data.table(
## 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 iris

rr = bmr$aggregate()[learner_id == "classif.featureless"]$resample_result[[1]]
rr$resampling$train_set(2)

---

**BenchmarkResult**

*Container for Benchmarking Results*

### Description

This is the result container object returned by `benchmark()`. A `BenchmarkResult` consists of the data row-bound data of multiple `ResampleResults`, which can be easily reconstructed.

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

### S3 Methods

- `as.data.table(bmr)`
  - `BenchmarkResult` -> `data.table::data.table()`
  - Returns a copy of the internal data.

- `c(...)`
  - `(BenchmarkResult, ...) -> BenchmarkResult`
  - Combines multiple objects convertible to `BenchmarkResult` into a new `BenchmarkResult`.

- `friedman.test(y,...)`
  - `BenchmarkResult` -> "htest"
  - Applies `friedman.test()` on the benchmark result, returning an object of class "htest".

### Public fields

- `data (data.table::data.table())`
  - Internal data storage with one row per resampling iteration. Can be joined with $rr_data by joining on column "hash". We discourage users to directly work with this table.
**BenchmarkResult**

`rr_data (data.table::data.table())`

Internal data storage with one row per ResampleResult (instead of one row per resampling iteration as in `$data`).

Package developers may opt to add additional columns here. These columns are preserved in all mutators.

Can be combined with `$data` by (left) joining on the key column "hash". E.g., `mlr3tuning` stores additional information for the optimization path in this table.

**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 NULL 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 getter, 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()`.

`resamplings (data.table::data.table())`

Table of included Resamplings with three columns:

- "resampling_hash" (character(1)),
- "resampling_id" (character(1)), and
- "resampling" (Resampling).

`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 = data.table())

*Arguments:*

data (**data.table::data.table()**)
Table with data for one resampling iteration per row, with at least the following columns:
• "task" (Task),
• "learner" (Learner),
• "resampling" (Resampling),
• "iteration" (integer(1)),
• "prediction" (Prediction), and
• "uhash" (character(1)).
Column "uhash" is the unique hash of the corresponding ResampleResult. Additional columns are kept in the resulting object, but otherwise ignored by BenchmarkResult.

**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 keeps 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)

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.

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 while discards all others. Same procedure for learner_ids and resampling_ids.

Usage:
BenchmarkResult$filter(
  task_ids = NULL,
  learner_ids = NULL,
  resampling_ids = NULL
)

Arguments:
task_ids (character())
  Ids of Tasks to keep.
learner_ids (character())
  Ids of Learners to keep.
resampling_ids (character())
  Ids 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 keeps 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.

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

bmr = benchmark(design)
print(bmr)

bmr$tasks
bmr$learners

# first 5 individual resamplings
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

---

**benchmark_grid**

*Generate a Benchmark Grid Design*

**Description**

Takes a list 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).
Value

\((\text{data.table}::\text{data.table()})\) with the cross product of the input vectors.

Examples

```r
tasks = list(tsk("iris"), tsk("sonar"))
learners = list(lrn("classif.featureless"), lrn("classif.rpart"))
resamplings = list(rsmp("cv"), rsmp("subsampling"))
benchmark_grid(tasks, learners, resamplings)
```

DataBackend

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.data.frame()`

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))
Description

DataBackend for data.table which serves as an efficient in-memory data base.

Super class

mlr3::DataBackend -> DataBackendDataTable

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

\textbf{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.

\textit{Usage}:
\texttt{DataBackendDataTable$data(rows, cols, data_format = "data.table")}

\textit{Arguments}:
rows integer()
    Row indices.
cols character()
    Column names.
data_format (\texttt{character(1)})
    Desired data format, e.g. "data.table" or "Matrix".

\textbf{Method head()}: Retrieve the first \texttt{n} rows.

\textit{Usage}:
\texttt{DataBackendDataTable$head(n = 6L)}

\textit{Arguments}:
\texttt{n} (\texttt{integer(1)})
    Number of rows.

\textit{Returns}: \texttt{data.table::data.table()} of the first \texttt{n} rows.

\textbf{Method distinct()}: Returns a named list of vectors of distinct values for each column specified. If \texttt{na.rm} is \texttt{TRUE}, missing values are removed from the returned vectors of distinct values. Non-existing rows and columns are silently ignored.

\textit{Usage}:
\texttt{DataBackendDataTable$distinct(rows, cols, na.rm = \texttt{TRUE})}

\textit{Arguments}:
rows integer()
    Row indices.
cols character()
    Column names.
na_rm logical(1)
    Whether to remove NAs or not.

\textit{Returns}: Named list() of distinct values.

\textbf{Method missings()}: Returns the number of missing values per column in the specified slice of data. Non-existing rows and columns are silently ignored.
DataBackendMatrix

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.data.frame()

Examples

data = as.data.table(iris)
data$id = seq_len(nrow(iris))
b = DataBackendDataTable$new(data = data, primary_key = "id")
print(b)
b/head()
b$data(rows = 100:101, cols = "Species")

b$nrow
head(b$rownames)

b$nrow
b$colnames

# alternative construction
as_data_backend(iris)

---

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:
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:
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.data.frame()
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**

```r
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**

```r
default_measures("classif")
default_measures("regr")
```
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() (soft 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 \texttt{mlr\_learners}, e.g. \texttt{classif\_rpart} or \texttt{regr\_rpart}.

More classification and regression learners are implemented in the add-on package \texttt{mlr3learners}. Learners for survival analysis (or more general, for probabilistic regression) can be found in \texttt{mlr3proba}.

The dictionary \texttt{mlr\_learners} gets automatically populated with the new learners as soon as the respective packages are loaded.

More (experimental) learners can be found on GitHub: \url{https://github.com/mlr3learners/}. A guide on how to extend \texttt{mlr3} with custom learners can be found in the \texttt{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:

- \texttt{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 \texttt{mlr3filters}.
- \texttt{selected\_features(...)}: Returns a subset of selected features as \texttt{character()}. The learner must be tagged with property "selected\_features".
- \texttt{oob\_error(...)}: Returns the out-of-bag error of the model as \texttt{numeric(1)}. The learner must be tagged with property "oob\_error".

Setting Hyperparameters

All information about hyperparameters is stored in the slot \texttt{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 \texttt{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"\}. 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.
fallback (Learner)
Learner which is fitted to impute predictions in case that either the model fitting or the prediction 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.

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()
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 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 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 keeps 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 $\text{train()}$ to create a new Prediction based on the new data in newdata. Object task is the task used during $\text{train()}$ and required for conversion of newdata. If the learner’s $\text{train()}$ 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 $\text{resample()}$ or $\text{benchmark()}$, you need to pass the corresponding task stored in the $\text{ResampleResult}$ or $\text{BenchmarkResult}$, respectively.

Usage:
Learner$\text{predict\_newdata(newdata, task = NULL)}$

Arguments:
newdata (data.frame())
  New data to predict on. Row ids are automatically created via auto-incrementing.
task (Task).

Returns: Prediction.

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

Usage:
Learner$\text{reset()}$

Returns: Returns the object itself, but modified by reference. You need to explicitly $\text{clone()}$ 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$\text{clone(deep = FALSE)}$

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.
### LearnerClassif

- "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`.

**Super class**

`mlr3::Learner` \(\rightarrow\) LearnerClassif

**Methods**

**Public methods:**

- `LearnerClassif$new()`
- `LearnerClassif$clone()`

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

**Usage:**

```r
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

# 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("iris")
lrn$train(task, 1:120)

# predict on new observations:
lrn$predict(task, 121:150)$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.

Predefined learners can be found in the dictionary mlr_learners. Essential regression learners can be found in this dictionary after loading mlr3learners.

Super class

mlr3::Learner -> LearnerRegr

Methods

Public methods:

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

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

Usage:

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().

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:
LearnerRegr$clone(deep = FALSE)

Arguments:
    deep  Whether to make a deep clone.

See Also

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

Examples

# 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")
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"]'). 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. Either "micro" or "macro".
- `aggregator` (function())
  Function to aggregate individual scores.
- `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:
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.
Measure

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"}. 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

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" 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().

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
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 R6 class.

Usage:
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\}. 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" and "se".

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"}. 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_coercions  Object Coercion
S3 generics and methods to coerce to (lists of) Task, Learner, Resampling, and Measure.
Usage

as_task(x, clone = FALSE)

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

as_tasks(x, clone = FALSE)

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

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

as_learner(x, clone = FALSE)

## 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)

as_resampling(x, clone = FALSE)

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

as_resamplings(x, clone = FALSE)

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

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

as_measure(x, task_type = NULL, clone = FALSE)

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

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

## S3 method for class `NULL`
as_measures(x, task_type = NULL, clone = FALSE)

## S3 method for class `list`
as_measures(x, task_type = NULL, clone = FALSE)

## S3 method for class `Measure`
as_measures(x, task_type = NULL, clone = FALSE)

Arguments

x (any)
Object to coerce.

clone (logical(1))
If TRUE, ensures that the returned object is not the same as the input x, e.g. by cloning it or constructing it from a dictionary such as mlr_learners.

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.

Value

Coerced object. The default method will return the object as-is. Failed coercions have to be handled by one of the assertions in mlr_assertions.

Examples

# convert single measure to list of measures
measure = msr("classif.ce")
as_measures(measure)

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 more classification and regression learners, load the mlr3learners package and https://github.com/mlr3learners.

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.

See Also

Sugar functions: lrn(), lrns()

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_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 raises an exception during train.
- **error_predict**: Probability to raise an exception during predict.
- **segfault_train**: Probability to provokes a segfault during train.
- **segfault_predict**: Probability to provokes 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.
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():

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

Super classes

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

Methods

Public methods:

- LearnerClassifDebug$new()
- LearnerClassifDebug$clone()

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

Usage:

```r
LearnerClassifDebug$new()
```

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

Usage:

```r
LearnerClassifDebug$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.featureless, mlr_learners_classif.rpart, mlr_learners_regr.featureless, mlr_learners_regr.rpart, mlr_learners
Examples

```r
learner = lrn("classif.debug")
learner$param_set$values = list(message_train = 1, save_tasks = TRUE)
# this should signal a message
task = tsk("iris")
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. Hyper-parameter 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()`:

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

Super classes

```r
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.

**Dictionary**

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

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

```r
lrn("classif.rpart")
```
Super classes

mlr3::Learner -> mlr3::LearnerClassif -> LearnerClassifRpart

Methods

Public methods:
- LearnerClassifRpart$new()
- LearnerClassifRpart$importance()
- LearnerClassifRpart$selected_features()
- LearnerClassifRpart$clone()

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

Usage:
LearnerClassifRpart$new()

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

Usage:
LearnerClassifRpart(importance)

Returns: Named numeric().

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

Usage:
LearnerClassifRpart$selected_features()

Returns: character().

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

Usage:
LearnerClassifRpart$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_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")
```

Super classes

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

Methods

Public 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

---

Regression Tree Learner

Description

A LearnerRegr for a regression tree implemented in rpart::rpart() in package rpart. Parameter xval is set to 0 in order to save some computation time.

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")

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.

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

See Also

Sugar functions: msr(), msrs()
Implementation of most measures: mlr3measures
Other Dictionary: mlr_learners, mlr_resamplings, mlr_task_generators, mlr_tasks
Other Measure: MeasureClassif, MeasureRegr, Measure, mlr_measures_classif.costs, mlr_measures_debug, mlr_measures_elapsed_time, mlr_measures_oob_error, mlr_measures_selected_features

Examples

as.data.table(mlr_measures)
mlr_measures$get("classif.ce")
msr("regr.mse")

mlr_measures_classif.acc

Classification Accuracy

Description

Classification 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():

mlr_measures$get("acc")
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 classification measures:

- `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.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`

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
mlr_measures$get("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

$$\frac{1}{\sum_i \hat{w}_i} \sum_i 1(r_i = t_i)\hat{w}_i.$$

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`

---

**mlr_measures_classif.bbrier**

*Binary Brier Score*

**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 m$sr()::

```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.


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.


mlr_measures_classif.costs

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?

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:**
- `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.mlreasures_debug, mlr_measures_elapsed_time, mlr_measures_oob_error, mlr_measures_selected_features, mlr_measures`


**Examples**

```r
# 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
```


```r
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

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

**Dictionary**

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

```r
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`.

**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.fbeta**

**F-beta Score**

**Description**

Binary classification measure defined with $P$ as `precision()` and $R$ as `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 `msr()`:

```r
mlr_measures$get("fbeta")
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`. 
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():

```
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.


Meta Information

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

Note

The score function calls \texttt{mlr3measures::fn()} 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: mlr_measures

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


---

\texttt{mlr_measures_classif.fnr}

\textit{False Negative Rate}

Description

Binary classification measure defined as

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

Also know 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.

Other classification measures:


Other binary classification measures:

False Omission Rate

Description

Binary classification measure defined as

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

Dictionary

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

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

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.

Other classification measures:

Other binary classification measures: \texttt{mlr\_measures\_classif.auc}, \texttt{mlr\_measures\_classif.bbrier}, \texttt{mlr\_measures\_classif.dor}, \texttt{mlr\_measures\_classif.fbeta}, \texttt{mlr\_measures\_classif.fdr}, \texttt{mlr\_measures\_classif.fn}, \texttt{mlr\_measures\_classif.fpr}, \texttt{mlr\_measures\_classif.fp}, \texttt{mlr\_measures\_classif.mcc}, \texttt{mlr\_measures\_classif.npv}, \texttt{mlr\_measures\_classif.ppv}, \texttt{mlr\_measures\_classif.precision}, \texttt{mlr\_measures\_classif.recall}, \texttt{mlr\_measures\_classif.sensitivity}, \texttt{mlr\_measures\_classif.specificity}, \texttt{mlr\_measures\_classif.tnr}, \texttt{mlr\_measures\_classif.tn}, \texttt{mlr\_measures\_classif.tpr}, \texttt{mlr\_measures\_classif.tp}

\begin{verbatim}
mlr_measures_classif.fp
False Positives
\end{verbatim}

\textbf{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.

\textbf{Dictionary}

This \texttt{Measure} can be instantiated via the dictionary \texttt{mlr\_measures} or with the associated sugar function \texttt{msr()}:\footnote{\texttt{mlr\_measures\$get("fp")} and \texttt{msr("fp")}}

\begin{verbatim}
mlr\_measures\$get("fp")
msr("fp")
\end{verbatim}

\textbf{Meta Information}

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

\textbf{Note}

The score function calls \texttt{mlr3\_measures::fp()} from package \texttt{mlr3\_measures}. If the measure is undefined for the input, \texttt{NaN} is returned. This can be customized by setting the field \texttt{na\_value}.

\textbf{See Also}

Dictionary of \texttt{Measures}: \texttt{mlr\_measures}

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

Other classification measures: \texttt{mlr\_measures\_classif.acc}, \texttt{mlr\_measures\_classif.auc}, \texttt{mlr\_measures\_classif.bacc}, \texttt{mlr\_measures\_classif.bbrier}, \texttt{mlr\_measures\_classif.ce}, \texttt{mlr\_measures\_classif.costs}, \texttt{mlr\_measures\_classif.dor}, \texttt{mlr\_measures\_classif.fbeta}, \texttt{mlr\_measures\_classif.fdr}, \texttt{mlr\_measures\_classif.fn}, \texttt{mlr\_measures\_classif.fomr}, \texttt{mlr\_measures\_classif.fpr}, \texttt{mlr\_measures\_classif.logloss}, \texttt{mlr\_measures\_classif.mcc}, \texttt{mlr\_measures\_classif.npv}, \texttt{mlr\_measures\_classif.ppv}, \texttt{mlr\_measures\_classif.precision}, \texttt{mlr\_measures\_classif.recall}, \texttt{mlr\_measures\_classif.sensitivity}, \texttt{mlr\_measures\_classif.specificity}, \texttt{mlr\_measures\_classif.tnr}, \texttt{mlr\_measures\_classif.tn}, \texttt{mlr\_measures\_classif.tpr}, \texttt{mlr\_measures\_classif.tp}
mlr_measures_classif.fpr

*False Positive Rate*

**Description**

Binary classification measure defined as

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

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()`:

```
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`. 
**mlr_measures_classif.logloss**

*Log Loss*

**Description**

Classification measure defined as

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

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()`:

```
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.


---

**mlr_measures_classif.mbrier**

*Multiclass Brier Score*

**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 `bbrier()`.

**Dictionary**

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

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

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

Note

The score function calls `mlr3measures::mbrier()` 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.fn`
- `mlr_measures_classif.fomr`
- `mlr_measures_classif.fpr`
- `mlr_measures_classif.fp`
- `mlr_measures_classif.logloss`
- `mlr_measures_classif.mcc`
- `mlr_measures_classif.npv`
- `mlr_measures_classif.ppv`
- `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`

Other multiclass classification measures:
- `mlr_measures_classif.acc`
- `mlr_measures_classif.bacc`
- `mlr_measures_classif.ce`
- `mlr_measures_classif.costs`
- `mlr_measures_classif.logloss`

Dictionary

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

```r
mlr_measures$set("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.


---

**mlr_measures_classif.npv**

*Negative Predictive Value*

Description

Binary classification measure defined as

\[
\frac{TN}{FN + TN}.
\]
Dictionary

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

```
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.


**mlr_measures_classif.ppv**

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

### Description

Binary classification measure defined as

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

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.
**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()`:

```
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.


---

**mlr_measures_classif.sensitivity**

*True Positive Rate*

**Description**

Binary classification measure defined as

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

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("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.


```
mlr_measures_classif.specificity
True Negative Rate
```

Description

Binary classification measure defined as

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

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.

Other classification measures:
True Negatives

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():

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

Meta Information

- Type: "binary"
- Range: [0, \(\infty\)]
- 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

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

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.

mlr_measures_classif.tp

True Positives

Description

Binary classification measure counting the true positives, i.e. the number of predictions correctly indicating a positive class label.

Dictionary

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

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

Meta Information

- Type: "binary"
- Range: [0, ∞)
- 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.
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("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.

---

**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: $[0, \infty)$
- Minimize: NA
- Required prediction: 'response'
Super class

\texttt{mlr3::Measure} -> \texttt{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:

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

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

Method \texttt{clone()}: The objects of this class are cloneable with this method.
  
  Usage:
  
  \texttt{MeasureDebug\$clone(deep = FALSE)}

Arguments:

deep Whether to make a deep clone.

See Also

Dictionary of Measures: \texttt{mlr\_measures}

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

implementations.

Other Measure: \texttt{Measure\_Classif,Measure\_Regr,Measure,mlr\_measures\_classif\_costs,mlr\_measures\_elapsed\_time,mlr\_measures\_oob\_error,mlr\_measures\_selected\_features,mlr\_measures}

Examples

\begin{verbatim}
  task = tsk("wine")
  learner = lrn("classif.featureless")
  measure = msr("debug")
  rr = resample(task, learner, rsmp("cv", folds = 3))
  rr$score(measure)
\end{verbatim}
**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()`:

```r
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**

`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:*

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

*Arguments:*

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

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

Meta Information

- Type: NA
- Range: (−∞, ∞)
- 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:

`MeasureOOBError$new()`

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

Usage:

`MeasureOOBError$clone(deep = FALSE)`

Arguments:

depth 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_selected_features`, `mlr_measures`

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()`:

`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.


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
**mlr_measures_regr.mae**

**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::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.
See Also

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


Description

Regression measure defined as
\[
\frac{1}{n} \sum_{i=1}^{n} \left| \frac{t_i - r_i}{t_i} \right|
\]

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.
**mlr_measures_regr.medae**

**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

\[
\text{median}_i |t_i - r_i|.
\]

**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

$$\text{medse} = \text{median}_{i} \left[(t_i - r_i)^2\right].$$

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.
### 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$get("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`.

---

**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():

```r
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.
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():

```r
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.
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.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():

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.


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():

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():

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.
**mle_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`.

**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.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.
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.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 \textit{rse()} from 1, hence it compares the squared error of the predictions relative to a naive model predicting the mean.

Dictionary

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

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

Meta Information

- Type: "regr"
- Range: \((-\infty, 1]\)
- Minimize: FALSE
- Required prediction: response

Note

The score function calls \texttt{mlr3measures::rsq()} 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: mlr_measures

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


---

**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():

```
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.
mlr_measures_regr.smape

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.smape

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 mlr3measures::smape() from package mlr3measures.

If the measure is undefined for the input, NaN is returned. This can be customized by setting the field na_value.
mnr_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 `mnr_measures` or with the associated sugar function `msr()`:

```r
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`.

**See Also**

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

Other regression measures: `mnr_measures_regr.bias, mnr_measures_regr.ktau, mnr_measures_regr.mae, mnr_measures_regr.mape, mnr_measures_regr.maxae, mnr_measures_regr.medae, mnr_measures_regr.medse, mnr_measures_regr.mse, mnr_measures_regr.msle, mnr_measures_regr.pbias, mnr_measures_regr.rae, mnr_measures_regr.rmse, mnr_measures_regr.rmsle, mnr_measures_regr.rrse, mnr_measures_regr.rsq, mnr_measures_regr.sae, mnr_measures_regr.srho, mnr_measures_regr.sse`
mlr_measures_regr.sse

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.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.
mnr_measures_selected_features

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():

mnr_measures$\text{get}("\text{selected\_features}\)"
mnr("\text{selected\_features}\)

Meta Information

- Type: NA
- Range: \([0, \infty)\)
- Minimize: TRUE
- Required prediction: \('\text{response}'\)

Super class

mnr3::Measure -> MeasureSelectedFeatures

Public fields

\texttt{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

---

Dictionary of Resampling Strategies

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.
S3 methods

- as.data.table(dict)
  
  \[
  \text{mlr3misc::Dictionary} \rightarrow \text{data.table::data.table()}
  \]
  
  Returns a \text{data.table::data.table()} with columns "key", "params", and "iters".

See Also

Sugar functions: \text{rsmp()}, \text{rsmps()}

Other Dictionary: \text{mlr_learners, mlr_measures, mlr_task_generators, mlr_tasks}

Other Resampling: \text{Resampling, mlr_resamplings_bootstrap, mlr_resamplings_custom, mlr_resamplings_cv, mlr_resamplings_holdout, mlr_resamplings_insample, mlr_resamplings_repeated_cv, mlr_resamplings_subsampling}

Examples

\[
\begin{align*}
\text{as.data.table(mlr_resamplings)} \\
\text{mlr_resamplings$get("cv")} \\
\text{rsmp("subsampling")}
\end{align*}
\]

\text{mldr_resamplings_bootstrap}

\text{Bootstrap Resampling}

Description

Splits data into bootstrap samples (sampling with replacement). Hyperparameters are the number of bootstrap iterations (\text{repeats}, default: 30) and the ratio of observations to draw per iteration (\text{ratio}, default: 1) for the training set.

Dictionary

This \text{Resampling} can be instantiated via the \text{dictionary mlr_resamplings} or with the associated sugar function \text{rsmp()}:

\[
\begin{align*}
\text{mlr_resamplings$get("bootstrap")} \\
\text{rsmp("bootstrap")}
\end{align*}
\]

Parameters

- \text{repeats (integer(1))}
  
  Number of repetitions.

- \text{ratio (numeric(1))}
  
  Ratio of observations to put into the training set.

Super class

\text{mlr3::Resampling} \rightarrow \text{ResamplingBootstrap}
Active bindings

- `iters (integer())`
  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:

`ResamplingBootstrap$new()`

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

Usage:

`ResamplingBootstrap$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_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("iris")
task$filter(1:10)

# Instantiate Resampling
rb = rsmp("bootstrap", repeats = 2, ratio = 1)
rb$instantiate(task)

# Individual sets:
rb$train_set(1)
rb$test_set(1)
intersect(rb$train_set(1), rb$test_set(1))

# Internal storage:
rb$instance$M # Matrix of counts

---

**mlr_resamplings_custom**

*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")
rsmp("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:*

```r
eachcaret$resamplings$custom
```
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_repeated_cv, mlr_resamplings_subsampling, mlr_resamplings`

Examples

```r
# Create a task with 10 observations
task = tsk("iris")
task$filter(1:10)

# Instantiate Resampling
rc = rsmp("custom")
train_sets = list(1:5, 5:10)
test_sets = list(5:10, 1:5)
rc$instantiate(task, train_sets, test_sets)

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():

```r
mlr_resamplings$get("bootstrap")
rsmp("bootstrap")
```

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:

```r
ResamplingCV$new()
```

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

Usage:

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

Arguments:

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

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_repeated_cv, mlr_resamplings_subsampling, mlr_resamplings

Examples
# Create a task with 10 observations
task = tsk("iris")
task$filter(1:10)

# Instantiate Resampling
rcv = rsmp("cv", folds = 3)
rsv$instance # table

# Individual sets:
rsv$train_set(1)
rsv$test_set(1)
intersect(rsv$train_set(1), rsv$test_set(1))

# Internal storage:
rsv$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:
  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_insample, mlr_resamplings_repeated_cv, mlr_resamplings_subsampling, mlr_resamplings
Examples

# Create a task with 10 observations
task = tsk("iris")
task$filter(1:10)

# Instantiate Resampling
rho = rsmp("holdout", ratio = 0.5)
rho$instance(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

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():

mlr_resamplings$get("insample")
rsmp("insample")

Super class

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:
ResamplingInsample$new()

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

Usage:
ResamplingInsample$clone(deep = FALSE)

Arguments:
deepp 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_repeated_cv, mlr_resamplings_subsampling, mlr_resamplings

Examples

# Create a task with 10 observations
task = tsk("iris")
task$filter(1:10)

# Instantiate Resampling
rins = rsmp("insample")
rins$instance(task)

rins$train_set(1)
rins$test_set(1)

# Internal storage:
rins$instance # just row ids

---

**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()`:

```r
mlr_resamplings$get("holdout")
rsmp("holdout")
```

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:

ResamplingRepeatedCV$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_subsampling, mlr_resamplings`

Examples

```
# Create a task with 10 observations
task = tsk("iris")
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
```
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:
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`

Examples

```r
# Create a task with 10 observations
task = tsk("iris")
task$filter(1:10)

# Instantiate Resampling
rss = rsmp("subsampling", repeats = 2, ratio = 0.5)
rss$instantiate(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`.
- `rsmp()` for a Resampling from `mlr_resamplings`.
mlr_sugar

- `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
tsks(.key, ...)
tgns(.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 `param::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
# iris task with new id
tsks("iris", id = "iris2")

# 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'
lapply(c("classif.featureless", "classif.rpart"), lrn, predict_type = "prob")

## mlr_tasks

### Dictionary of 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.

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)`

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

  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_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo

#### Examples

```r
as.data.table(mlr_tasks)
task = mlr_tasks$get("iris") # same as tsk("iris")
head(task$data())

# Add a new task, based on a subset of iris:
data = iris
```
data$Species = factor(ifelse(data$Species == "setosa", "1", "0"))
task = TaskClassif$new("iris.binary", data, target = "Species", positive = "1")

# add to dictionary
mlr_tasks$add("iris.binary", task)

# list available tasks
mlr_tasks$keys()

# retrieve from dictionary
mlr_tasks$get("iris.binary")

# remove task again
mlr_tasks$remove("iris.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**

mlr_tasks$get("boston_housing")
tsk("boston_housing")

**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_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
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

mlr_tasks$get("german_credit")
tsk("german_credit")

Source

Data set originally published on UCI. This is the preprocessed version taken from package evtree.

Donor: Professor Dr. Hans Hofmann
Institut für Statistik und Ökonometrie
Universität Hamburg
FB Wirtschaftswissenschaften
Von-Melle-Park 5
2000 Hamburg 13

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_iris, mlr_tasks_mtcars, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo, mlr_tasks

Examples

task = tsk("german_credit")
costs = matrix(c(0, 1, 5, 0), nrow = 2)
dimnames(costs) = list(predicted = task$class_names, truth = task$class_names)
measure = msr("classif.costs", id = "german_credit_costs", costs = costs)
print(measure)
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")

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_mtcars, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo, mlr_tasks

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
mlr_tasks$get("mtcars")
tsk("mtcars")
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_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")

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_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks_zoo, mlr_tasks

---

**mlr_tasks_sonar**  
*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.
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")
  tsk("spam")

Source
Donor: George Forman (gforman at nospm hpl.hp.com) 650-857-7835
Preprocessing: Columns have been renamed. Preprocessed data taken from the kernlab package.

References
mrl_tasks_wine

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_pima, mlr_tasks_sonar, mlr_tasks_wine, mlr_tasks_zoo, mlr_tasks

mrl_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”)

Source

Donor: Stefan Aeberhard, email: stefan@coral.cs.jcu.edu.au

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_german_credit, mlr_tasks_iris, mlr_tasks_mtcars, mlr_tasks_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_zoo, mlr_tasks
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

```r
mlr_tasks$get("zoo")
tsk("zoo")
```

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_pima, mlr_tasks_sonar, mlr_tasks_spam, mlr_tasks_wine, mlr_tasks`

---

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_friedman1`, `mlr_task_generators_smiley`, `mlr_task_generators_xor`

Examples

```r
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()`:

```r
mlr_task_generators$get("2dnormals")
tgen("2dnormals")
```

### Super class

`mlr3::TaskGenerator` -> `TaskGenerator2DNormals`

### Methods

**Public methods:**

- `TaskGenerator2DNormals$new()`
- `TaskGenerator2DNormals$clone()`

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

Usage:

```r
TaskGenerator2DNormals$new()
```
Method clone(): The objects of this class are cloneable with this method.

Usage:
TaskGenerator2DNormals$clone(deep = FALSE)

Arguments:
depth 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_friedman1, mlr_task_generators_smiley, mlr_task_generators_xor, mlr_task_generators

Examples
tgen("2dnormals")$generate(10)$data()

---

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:**
depth 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_smiley, mlr_task_generators_xor, mlr_task_generators

Examples

tgen("friedman1")$generate(10)$data()

---

**mlr_task_generators_smiley**

_Smiley Classification Task Generator_

**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
mlr_task_generators$get("smiley")
tgen("smiley")
```

**Super class**

mlr3::TaskGenerator -> TaskGeneratorSmiley
**mlr_task_generators_xor**

### Methods

**Public methods:**

- `TaskGeneratorSmiley$new()`
- `TaskGeneratorSmiley$clone()`

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

*Usage:*

`TaskGeneratorSmiley$new()`

**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_friedman1, mlr_task_generators_xor, mlr_task_generators`

### Examples

```r
  tgen("smiley")$generate(10)$data()
```

---

**mlr_task_generators_xor**

**XOR Classification Task Generator**

### 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
  mlr_task_generators$get("xor")
  tgen("xor")
```

### Super class

`mlr3::TaskGenerator` -> `TaskGeneratorXor`
predict.Learner

Methods

Public methods:

- TaskGeneratorXor$new()
- TaskGeneratorXor$clone()

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

Usage:
TaskGeneratorXor$new()

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

Usage:
TaskGeneratorXor$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_taskGenerators_friedman1, mlr_taskGenerators_smiley, mlr_taskGenerators

Examples

tgen("xor")$generate(10)$data()

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

## S3 method for class 'Learner'
predict(object, newdata, predict_type = NULL, ...)
Prediction

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.

Examples

```r
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>")
```

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().
Prediction

- \( \text{c(..., keep_duplicates = TRUE)} \)
  \( \text{(Prediction, Prediction, ...)} \rightarrow \text{Prediction} \)
  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(1))
  Required properties of the Task.

(predict_types (character(1))
  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(1))
  Vector of row ids for which predictions are stored.

(truth (any)
  True (observed) outcome.

(missing (integer(1))
  Returns row_ids for which the predictions are missing or incomplete.

Methods

Public methods:
- \text{Prediction$format()}
- \text{Prediction$print()}
- \text{Prediction$help()}
- \text{Prediction$score()}
- \text{Prediction$clone()}

Method format(): Helper for print outputs.
  Usage:
  \text{Prediction$format()}

Method print(): Printer.
  Usage:
  \text{Prediction$print(...)}
**PredictionClassif**  

**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.

**Usage:**

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`

---

**PredictionClassif**  

**Prediction Object for Classification**

**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.
missing (integer())
Returns row_ids for which the predictions are missing or incomplete.

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
)

Arguments:

  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".

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

```r
task = tsk("iris")
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"))
```

---

**PredictionRegr**  
*Prediction Object for Regression*

**Description**

This object wraps the predictions returned by a learner of class **LearnerRegr**, i.e. the predicted response and standard error.

**Super class**

`mlr3::Prediction` -> **PredictionRegr**

**Active bindings**

- `response` (numeric()): Access the stored predicted response.
- `se` (numeric()): Access the stored standard error.
- `missing` (integer()): Returns row_ids for which the predictions are missing or incomplete.

**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
)

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.

See Also

Other Prediction: PredictionClassif, Prediction

Examples

```r
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.

Usage

`resample(task, learner, resampling, store_models = FALSE)`
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.

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 back-end; 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 computed in order to reduce memory consumption. If you need access to the models for later analysis, set store_models to TRUE.
### Examples

```r
task = tsk("iris")
learner = lrn("classif.rpart")
resampling = rsmp("cv")

# 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)`
  - `ResampleResult` -> `data.table::data.table()`
  - Returns a copy of the internal data.
- `c(...)`
  - `(ResampleResult, ...) -> BenchmarkResult`
  - Combines multiple objects convertible to `BenchmarkResult` into a new `BenchmarkResult`.
ResampleResult

Public fields

data (data.table::data.table())
Internal data storage. We discourage users to directly work with this field.

Active bindings

task (Task)
The task resample() operated on.

learners (list of Learner)
List of trained learners, sorted by resampling iteration.

resampling (Resampling)
Instantiated Resampling object which stores the splits into training and test.

uhash (character(1))
Unique hash for this object.

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.

errors (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.

Usage:
ResampleResult$new(data, uhash = NULL)

Arguments:

data (data.table::data.table())
Table with data for one resampling iteration per row: Task, Learner, Resampling, iteration (integer(1)), and Prediction.

uhash (character(1))
Unique hash for this ResampleResult. If NULL, a new unique hash is generated. This unique hash is primarily needed to group information in BenchmarkResults.
Method format(): Helper for print outputs.
  Usage:
  ResampleResult$format()

Method print(): Printer.
  Usage:
  ResampleResult$print()
  Arguments:
  ... (ignored).

Method help(): Opens the corresponding help page referenced by field $man.
  Usage:
  ResampleResult$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:
  ResampleResult$prediction(predict_sets = "test")
  Arguments:
  predict_sets (character())
    Subset of ["train", "test"].
  Returns: Prediction.

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)
  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.

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 keeps 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("iris")
learner = lrn("classif.rpart")
resampling = rsmp("cv", folds = 3)
rr = resample(task, learner, resampling)
print(rr)

rr$aggregate(msr("classif.acc"))
rr$prediction()
rr$prediction()$confusion
rr$warnings
rr$errors
Resampling

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 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 forBootstrap, 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 (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.
Resampling

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_insample, mlr_resamplings_repeated_cv, mlr_resamplings_subsampling

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 iris task
task = tsk("iris")
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

```r
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
```

---

## Task

### Task Class

This is the abstract base class for task objects like `TaskClassif` and `TaskRegr`.

Tasks serve two purposes:

1. Tasks wrap a `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 `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 `mlr_tasks`, e.g. `iris` or `boston_housing`.

### S3 methods

- `as.data.table(t)`
  Task -> `data.table::data.table()`
  Returns the complete data as `data.table::data.table()`.

### Task mutators

The following methods change the task in-place:

- Any modification to `$col_roles` and `$row_roles`. This provides a different "view" on the data without altering the data itself.
- `$filter()` and `$select()` subset the set of active rows or features in `$row_roles` or `$col_roles`, respectively. This provides a different "view" on the data without altering the data itself.
- `$rbind()` and `$cbind()` change the task in-place by binding rows or columns to the data, but without modifying the original `DataBackend`. Instead, the methods first create a new `DataBackendDataTable` from the provided new data, and then merge both backends into an abstract `DataBackend` which merges the results on-demand.
- `rename()` wraps the `DataBackend` of the Task in an additional `DataBackend` which deals with the renaming. Also updates `$col_roles` and `$col_info`. 
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`.

- **backend** (DataBackend)
  Abstract interface to the data of the task.

- **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 `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.

- **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.

- **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".

- **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": Hold the observations back unless explicitly requested. Validation sets are not yet completely integrated into the package.

`row_roles` keeps track of the roles with a named list, 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". They will be exclusively assigned to be either in the training set or in the test set for each resampling iteration. 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.

`col_roles` keeps track of the roles with a named list, the 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(), ...).

`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", 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.

weights (\texttt{data.table::data.table()})
If the task has a column with designated role "weight", table with two columns:
- row_id (\texttt{integer()})
- observation weights weight (\texttt{numeric()}).
Returns NULL if there are is no weight column.

Methods

Public methods:
- Task$\texttt{new}() 
- Task$\texttt{help}() 
- Task$\texttt{format}() 
- Task$\texttt{print}() 
- Task$\texttt{data}() 
- Task$\texttt{formula}() 
- Task$\texttt{head}() 
- Task$\texttt{levels}() 
- Task$\texttt{missings}() 
- Task$\texttt{filter}() 
- Task$\texttt{select}() 
- Task$\texttt{rbind}() 
- Task$\texttt{cbind}() 
- Task$\texttt{rename}() 
- Task$\texttt{set\_row\_role}() 
- Task$\texttt{set\_col\_role}() 
- Task$\texttt{droplevels}() 
- Task$\texttt{clone}()

Method \texttt{new}(): Creates a new instance of this \texttt{R6} class.
Note that this object is typically constructed via a derived classes, e.g. \texttt{TaskClassif} or \texttt{TaskRegr}.

Usage:
Task$\texttt{new(id, task\_type, backend)}

Arguments:
id (\texttt{character(1)})
  Identifier for the new instance.
task\_type (\texttt{character(1)})
  Type of task, e.g. "regr" or "classif". Must be an element of \texttt{mlr\_reflections$task\_types$\texttt{type}.}
backend (\texttt{DataBackend})
  Either a \texttt{DataBackend}, or any object which is convertible to a \texttt{DataBackend} with \texttt{as\_data\_backend()}. E.g., a \texttt{data.frame()} will be converted to a \texttt{DataBackend\_DataTable}.

Method \texttt{help}(): Opens the corresponding help page referenced by field $\texttt{man}.
Usage:
Task$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 are additionally subsetted to only contain observations with role "use", and columns are filtered to only contain features with roles "target" and "feature". If invalid `rows` or `cols` are specified, an exception is raised.

Usage:
Task$data(rows = NULL, cols = NULL, 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".

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$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 keeps 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:
Task$select(cols)

Arguments:
cols character()
   Column names.
Task

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:
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:
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.

Usage:
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_role(): Adds the roles new_roles to rows referred to by row ids rows. If exclusive is TRUE, the referenced rows will be removed from all other roles.
This function is deprecated and will be removed in the next version in favor of directly modifying $row_roles.
Usage:
Task$set_row_role(rows, new_roles, exclusive = TRUE)

Arguments:
rows integer()
  Row indices.
new_roles (character()).
exclusive (logical(1)).

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 set_col_role(): Adds the roles new_roles to columns referred to by column names cols. If exclusive is TRUE, the referenced columns will be removed from all other roles. This function is deprecated and will be removed in the next version in favor of directly modifying $col_roles.

Usage:
Task$set_col_role(cols, new_roles, exclusive = TRUE)

Arguments:
cols character()
  Column names.
new_roles (character()).
exclusive (logical(1)).

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 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
Other Task: TaskClassif, TaskRegr, TaskSupervised, TaskUnsupervised, mlr_tasks_boston_housing, 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
Examples

```r
# we use the inherited class TaskClassif here, 
# Class Task is not intended for direct use 
task = TaskClassif$new("iris", iris, target = "Species")

task$nrow
task$ncol
task$feature_names
task$formula()

# de-select "Petal.Width"
task$select(setdiff(task$feature_names, "Petal.Width"))

task$feature_names

# Add new column "foo"
task$cbind(data.frame(foo = 1:150))
task$head()
```

---

**TaskClassif**  
*Classification Task*

---

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.

Super classes

```r
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.

*Usage:*

```r
task_classif$new(id, backend, target, positive = NULL)
```

*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.

Method `data()`: Calls $data from parent class Task and ensures that levels of the target column are in the right order.

*Usage:*

```r
task_classif$data(rows = NULL, cols = NULL, 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”.

*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:*

```r
task_classif$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:
deep Whether to make a deep clone.

See Also
Other Task: TaskRegr, TaskSupervised, TaskUnsupervised, Task, mlr_tasks_boston_housing,
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

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:
mblr_reflections$task_properties$classif

---

**TaskGenerator**

**TaskGenerator Class**

**Description**

Creates a Task of arbitrary size. Predefined task generators are stored in the dictionary mlr_task_generators, e.g. xor.
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.

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 [pkg][topic] pointing to a manual page for this object. Defaults to NA, but can be set by child classes.

Methods

Public methods:
  • TaskGenerator$new()
  • TaskGenerator$generate()
  • TaskGenerator$clone()

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

Usage:
TaskGenerator$new(
  id,             
  task_type,      
  packages = character(),
  param_set = ParamSet$new(),
  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.

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().

param_set (paradox::ParamSet)
  Set of hyperparameters.

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 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.

---

## 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.

### Super classes

dlrv::Task -> mlr3::TaskSupervised -> TaskRegr

### Methods

**Public methods:**
- TaskRegr$new()
- TaskRegr$truth()
- TaskRegr$clone()

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

**Usage:**
TaskRegr$new(id, backend, target)

**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.

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_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

Examples
  task = TaskRegr$new("iris", backend = iris, target = "Sepal.Length")
  task$task_type
  task$formula()
  task$truth()

  # possible properties:
  mlr_reflections$task_properties$regr
Index

Topic datasets
  mlr_learners, 43
  mlr_measures, 51
  mlr_resamplings, 109
  mlr_task_generators, 131
  mlr_tasks, 124

as_benchmark_result, 6
as_data_backend
  (as_data_backend.data.frame), 7
as_data_backend(), 17
as_data_backend.data.frame, 7, 17, 20, 22
as_learner (mlr_coercions), 41
as_learners (mlr_coercions), 41
as_measure (mlr_coercions), 41
as_measures (mlr_coercions), 41
as_resampling (mlr_coercions), 41
as_resamplings (mlr_coercions), 41
as_task (mlr_coercions), 41
as_tasks (mlr_coercions), 41

bbrier(), 70
benchmark, 8
benchmark(), 10, 25, 29, 34, 36, 37, 39, 41, 149
benchmark_grid, 15
benchmark_grid(), 8
BenchmarkResult, 6, 8, 10, 11, 12, 25, 29, 34, 145, 146
bootstrap, 149
boston_housing, 153

c(), 12
classif.auc, 34
classif.ce, 39
classif.rpart, 24
cv, 149

data.frame(), 7, 8, 21, 137
data.table(), 19
data.table::as.data.table(), 7
data.table::copy(), 18
data.table::data.table(), 7, 8, 10–13, 16, 17, 19, 21, 22, 26, 44, 52, 110, 124, 132, 137, 145, 146, 148, 153–158, 162
DataBackend, 7, 16, 18, 20, 22, 153, 154, 156–159, 162, 166
DataBackendDataTable, 7, 16, 17, 18, 22, 153, 156, 162, 166
DataBackendMatrix, 7, 16, 17, 20, 20
datasets::iris, 127
datasets::mtcars, 127
default_measures, 23
default_measures(), 147, 148

expand.grid(), 15
formula(), 157
friedman.test(), 10
future::plan(), 8, 144

intersect(), 155
iris, 153

LearnerClassif, 24, 27, 29, 33, 44–48, 50, 51, 139
 LearnerClassifDebug
  (mlr_learners_classif.debug), 44
 LearnerClassifFeatureless
  (mlr_learners_classif.featureless), 46
 LearnerClassifRpart
  (mlr_learners_classif.rpart), 47
 LearnerRegr, 24, 27, 29, 31, 32, 44, 45, 47–51, 142
 LearnerRegrFeatureless
  (mlr_learners_regr.featureless), 49
 LearnerRegrRpart
  (mlr_learners_regr.rpart), 50
 Learners, 45, 47, 48, 50, 51
 lgr::lgr-package, 9, 144
 lrn (mlr_sugar), 122
 lrn(), 43–47, 49, 50
 lrns (mlr_sugar), 122
 lrns(), 43, 44
 mad(), 49
 Matrix::Matrix(), 17, 21
 Matrix::sparseMatrix(), 16
 max.col(), 141
 mbrier(), 55
 mean(), 36, 39, 41, 49
 Measure, 13, 23, 25, 34, 34, 38–41, 51–109, 123, 139, 147, 148
 MeasureClassif, 34, 35, 37, 38, 41, 52, 59, 85, 87, 88, 109
 MeasureClassifCosts, 126
 MeasureClassifCosts
  (mlr_measures_classif.costs), 58
 MeasureDebug (mlr_measures_debug), 84
 MeasureElapsedTime
  (mlr_measures_elapsed_time), 86
 MeasureOOBEError
  (mlr_measures_oob_error), 87
 MeasureRegr, 34, 35, 37, 39, 52, 59, 85, 87, 88, 109
 MeasureSelectedFeatures
  (mlr_measures_selected_features), 108
 median(), 49
 mlbench::BostonHousing2, 125
 mlbench::mlbench.2dnormals(), 132
 mlbench::mlbench.friedman1(), 133
 mlbench::mlbench.smiley(), 134
 mlbench::mlbench.xor(), 135
 mlbench::PimaIndiansDiabetes2, 128
 mlbench::Sonar, 128
 mlbench::Zoo, 131
 mlr3 (mlr3-package), 5
 mlr3-package, 5
 mlr3::DataBackend, 18, 20
 mlr3::Learner, 30, 32, 45, 46, 48–50
 mlr3::LearnerClassif, 45, 46, 48
 mlr3::LearnerRegr, 49, 50
 mlr3::Measure, 38, 40, 58, 85–87, 108
 mlr3::MeasureClassif, 58
 mlr3::Prediction, 140, 142
 mlr3::Resampling, 110, 112, 114, 116, 117, 119, 121
 mlr3::Task, 161, 165
 mlr3::TaskGenerator, 132–135
 mlr3::TaskSupervised, 161, 165
 mlr3measures::acc(), 53
 mlr3measures::auc(), 54
 mlr3measures::bacc(), 55
 mlr3measures::bbrier(), 56
 mlr3measures::bias(), 89
 mlr3measures::ce(), 57
 mlr3measures::dor(), 60
 mlr3measures::fbeta(), 61
 mlr3measures::fdr(), 63
 mlr3measures::fn(), 64
 mlr3measures::fmr(), 65
 mlr3measures::fmr(), 66
 mlr3measures::fp(), 67
 mlr3measures::fpr(), 68
 mlr3measures::ktau(), 90
 mlr3measures::logloss(), 70
 mlr3measures::mae(), 90
 mlr3measures::mape(), 91
 mlr3measures::maxae(), 92
 mlr3measures::mbrier(), 71
 mlr3measures::mcc(), 72
 mlr3measures::medae(), 93
 mlr3measures::medse(), 94
 mlr3measures::mse(), 95
 mlr3measures::msle(), 96
mlr3measures::npv(). 73
mlr3measures::pbias(). 97
mlr3measures::ppv(). 74
mlr3measures::precision(). 75
mlr3measures::rae(). 98
mlr3measures::recall(). 77
mlr3measures::rmse(). 99
mlr3measures::rmsle(). 100
mlr3measures::rrse(). 101
mlr3measures::rse(). 102
mlr3measures::rsq(). 103
mlr3measures::sae(). 104
mlr3measures::sensitivity(). 78
mlr3measures::specificity(). 79
mlr3measures::srho(). 106
mlr3measures::sse(). 107
mlr3measures::tn(). 80
mlr3measures::tnr(). 81
mlr3measures::tp(). 82
mlr3measures::tpr(). 84
mlr3misc::Dictionary. 43, 44, 51, 52, 109, 110, 122, 124, 131, 132
mlr3misc::dictionary_sugar_get(). 122, 123
mlr3misc::encapsulate(). 26
mlr3misc::insert_named(). 25
mlr3misc::unnest(). 13
mlr Assertions. 43
mlr_coercions. 41
mlr_learners. 24, 29–33, 43, 45–52, 110, 122, 124, 132
mlr_learners_classif.debug. 29, 31, 33, 44, 47, 48, 50, 51
mlr_learners_classif.featureless. 29, 31, 33, 44, 45, 46, 48, 50, 51
mlr_learners_classif.rpart. 29, 31, 33, 44, 45, 47, 48, 50, 51
mlr_learners_regr.featureless. 29, 31, 33, 44, 45, 47, 48, 49, 51
mlr_learners_regr.rpart. 29, 31, 33, 44, 45, 47, 48, 50, 51
mlr_measures. 34, 37–41, 44, 51, 52–110, 123, 124, 132
mlr_measures_classif.auc. 53, 55–57, 59–84
mlr_measures_classif.bbrier. 53–55, 55, 57, 59–84
mlr_measures_classif.dor. 53–57, 59, 60, 62–80, 82–84
mlr_measures_classif.fbeta. 53–57, 59, 61, 61, 63–80, 82–84
mlr_measures_classif.fdr. 53–57, 59, 61, 62, 64–80, 82–84
mlr_measures_classif.fn. 53–57, 59, 61–63, 63, 65–84
mlr_measures_classif.fnr. 53–57, 59, 61–64, 66–80, 82–84
mlr_measures_classif.fomr. 53–57, 59, 61–65, 66, 67–84
mlr_measures_classif.fpr. 53–57, 59, 61–68, 68, 70–84
mlr_measures_classif.mbrier. 53–57, 59, 61–66, 68–70, 70, 72–74, 76–80, 82–84
mlr_measures_classif.mcc. 53–57, 59, 61–71, 71, 73–84
mlr_measures_classif.npv. 53–57, 59, 61–72, 72, 74–84
mlr_measures_classif.ppv. 53–57, 59, 61–73, 74, 76–84
mlr_measures_classif.precision. 53–57, 59, 59–75, 75, 77–84
mlr_measures_classif.recall. 53–57, 59, 61–76, 76, 78–84
mlr_measures_classif.sensitivity. 53–57, 59, 61–77, 77, 79–84
mlr_measures_classif.specificity. 53–57, 59, 61–78, 78, 80–84
mlr_measures_classif.tn. 53–57, 59, 61–79, 80, 82–84
mlr_measures_classif.tnr. 53–57, 59,
mlr_measures_classif.tp, 53–57, 59, 61–82, 84
mlr_measures_classif.tpr, 53–57, 59, 61–83, 83
mlr_measures_debug, 37, 39, 41, 52, 59, 84, 87, 88, 109
mlr_measures_elapsed_time, 37, 39, 41, 52, 59, 85, 86, 88, 109
mlr_measures_oob_error, 37, 39, 41, 52, 59, 85, 86, 87, 109
mlr_measures_regr.bias, 88, 90–108
mlr_measures_regr.ktau, 89, 89, 91–108
mlr_measures_regr.mape, 89–91, 91, 93–108
mlr_measures_regr.maxae, 89–92, 92, 94–108
mlr_measures_regr.medae, 89–93, 93, 95–108
mlr_measures_regr.medse, 89–94, 94, 96–108
mlr_measures_regr.mse, 89–95, 95, 97–108
mlr_measures_regr.msle, 89–96, 96, 98–108
mlr_measures_regr.pbias, 89–97, 97, 99–108
mlr_measures_regr.rae, 89–98, 98, 100–108
mlr_measures_regr.rmsle, 89–100, 100, 102–108
mlr_measures_regr.rrse, 89–101, 101, 103–108
mlr_measures_regr.rse, 89–102, 102, 104–108
mlr_measures_regr.rsq, 89–103, 103, 105–108
mlr_measures_regr.sae, 89–104, 104, 106–108
mlr_measures_regr.smape, 89–105, 105, 107, 108
mlr_measures_regr.srho, 89–106, 106, 108
mlr_measures_regr.sse, 89–107, 107
mlr_measures_selected_features, 37, 39, 41, 52, 59, 85, 87, 88, 108
mlr_measures_time_both
(mlr_measures_elapsed_time), 86
mlr_measures_time_predict
(mlr_measures_elapsed_time), 86
mlr_measures_time_train
(mlr_measures_elapsed_time), 86
mlr_reflections, 137
mlr_reflections$default_measures, 23, 43
mlr_reflections$learner_predict_types,
25, 27, 30, 33, 36, 39, 41
mlr_reflections$learner_properties, 25, 27, 30, 33
mlr_reflections$measure_properties, 36, 39, 41
mlr_reflections$task_feature_types, 25, 27, 30, 33, 154
mlr_reflections$task_properties, 154
mlr_reflections$task_types$type, 25, 27, 34, 35, 154, 156, 164
ml_resamplings, 44, 52, 109, 110–124, 132, 149, 152
ml_resamplings_bootstrap, 110, 110, 113, 115, 116, 118, 120, 122, 152
ml_resamplings_custom, 110, 111, 112, 115, 116, 118, 120, 122, 152
ml_resamplings_cv, 110, 111, 113, 114, 116, 118, 120, 122, 152
ml_resamplings_holdout, 110, 111, 113, 115, 115, 118, 120, 122, 152
ml_resamplings_insample, 110, 111, 113, 115, 116, 117, 120, 122, 152
ml_resamplings_repeated_cv, 110, 111, 113, 115, 116, 118, 120, 122, 152
ml_resamplings_subsampling, 110, 111, 113, 115, 115, 116, 118, 120, 121, 152
ml_sugar, 122
mlr_task_generators, 44, 52, 110, 122, 124, 131, 132–136, 163, 165
mlr_task_generators_2dnormals, 132, 132, 134–136, 165
mlr_task_generators_friedman1, 132, 133, 133, 135, 136, 165
mlr_task_generators_smiley, 132–134, 134, 136, 165
mlr_task_generators_xor, 132–135, 135, 165
ml_tasks, 44, 52, 110, 122, 124, 125–132, 153, 160, 161, 163, 165, 166
mlr_tasks_boston_housing, 124, 125, 126–131, 160, 163, 166
mlr_tasks_german_credit, 124, 125, 126, 127–131, 160, 163, 166
mlr_tasks_iris, 124–126, 127, 128–131, 160, 163, 166
mlr_tasks_sonar, 124–128, 128, 130, 131, 160, 163, 166
mlr_tasks_spam, 124–129, 129, 130, 131, 160, 163, 166
mlr_tasks_wine, 124–130, 130, 131, 160, 163, 166
mlr_tasks_zoo, 124–130, 131, 160, 163, 166
msr (mlr_sugar), 122
msrs (mlr_sugar), 122
msrs(), 51, 52
paradox::ParamSet, 24, 26, 27, 30, 32, 123, 149, 151, 164
ParamSet, 25
precision(), 61
predict.Learner, 136
Prediction, 11–13, 25, 28, 29, 32, 34, 36, 37, 39, 40, 84, 136, 137, 137, 138, 139, 141, 143, 146, 147
PredictionClassif, 29, 137, 139, 143
PredictionRegr, 32, 137, 139, 141, 142
progressr::with_progress(), 8, 144
R6, 12, 17, 18, 21, 27, 30, 32, 35, 38, 40, 45, 47–50, 59, 85, 86, 88, 109, 111, 112, 114, 116, 118, 119, 121, 132, 133, 135, 136, 140, 142, 146, 150, 156, 162, 164, 165
R6::R6Class, 43, 51, 109, 123–131
recall(), 61
regr.mse, 41
regr.rpart, 24
requireNamespace(), 28, 31, 33, 36, 39, 41, 164
resample, 143
resample(), 25, 29, 34, 36, 37, 39, 41, 145, 146, 149
ResampleResult, 6, 10–14, 25, 29, 34, 36, 37, 39–41, 144, 145, 145, 148
Resampling, 8, 11–15, 36, 39, 41, 109–123, 143, 144, 146, 147, 149, 150, 152, 155, 156
ResamplingBootstrap, 149
ResamplingBootstrap
  (mlr_resamplings_bootstrap), 110
ResamplingCustom
  (mlr_resamplings_custom), 112
ResamplingCV, 149, 151
ResamplingCV (mlr_resamplings_cv), 114
ResamplingHoldout, 151
ResamplingHoldout
  (mlr_resamplings_holdout), 115
ResamplingInsample
  (mlr_resamplings_insample), 117
ResamplingRepeatedCV
  (mlr_resamplings_repeated_cv), 118
Resamplings, 111, 113, 115, 116, 118, 120, 122, 152
ResamplingSubsampling
  (mlr_resamplings_subsampling), 121
rpart::rpart(), 47, 50
rse(), 103
rsmp (mlr_sugar), 122
rsmp(), 109, 110, 112, 114, 115, 117, 119, 121
rsmps (mlr_sugar), 122
rsmps(), 109, 110
sd(), 49
setdiff(), 155
stats::cor(), 89, 106
stats::predict(), 136
TaskClassif, 124–131, 141, 153, 156, 160, 161, 166
TaskGenerator, 122, 131–136, 163
TaskGenerator2DNormals
  (mlr_task_generators_2dnormals), 132
TaskGeneratorFriedman1
(mlr_task_generators_friedman1), 133
TaskGenerators, 133–136
TaskGeneratorSmiley
(mlr_task_generators_smiley), 134
TaskGeneratorXor
(mlr_task_generators_xor), 135
TaskRegr, 124–131, 143, 153, 156, 160, 163, 165
Tasks, 125–131
TaskSupervised, 124–131, 160, 161, 163, 165, 166
TaskUnsupervised, 124–131, 160, 163, 166
tgen (mlr_sugar), 122
tgen(), 131–135
tgens (mlr_sugar), 122
tgens(), 131, 132
time_train, 34
tsk (mlr_sugar), 122
tsk(), 124
tsks (mlr_sugar), 122
tsks(), 124
union(), 155
xor, 163