Package ‘mlr3’

December 19, 2019

Title Machine Learning in R - Next Generation

Version 0.1.6

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 backports, checkmate (>= 1.9.3), data.table, digest, lgr (>= 0.3.0), mlbench, mlr3measures (>= 0.1.1), mlr3misc (>= 0.1.6), paradox, R6, uuid

Suggests bibtex, callr, datasets, evaluate, future (>= 1.9.0), future.apply (>= 1.1.0), future.callr, Matrix, rpart, testthat, titanic

RdMacros mlr3misc

Encoding UTF-8

LazyData true

NeedsCompilation no

RoxygenNote 7.0.2

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'
- 'MeasureOObError.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'
- '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' 'Task_mutators.R' 'as_data_backend.R'
- 'assertions.R' 'benchmark.R' 'benchmark_grid.R'
- 'default_measures.R' 'deprecated.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  2019-12-19 08:50:02 UTC
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LearnerClassif</td>
<td>24</td>
</tr>
<tr>
<td>LearnerRegr</td>
<td>25</td>
</tr>
<tr>
<td>Measure</td>
<td>26</td>
</tr>
<tr>
<td>MeasureClassif</td>
<td>28</td>
</tr>
<tr>
<td>MeasureRegr</td>
<td>29</td>
</tr>
<tr>
<td>mlr_learners</td>
<td>30</td>
</tr>
<tr>
<td>mlr_learners_classif.debug</td>
<td>31</td>
</tr>
<tr>
<td>mlr_learners_classif.featureless</td>
<td>32</td>
</tr>
<tr>
<td>mlr_learners_classif.rpart</td>
<td>33</td>
</tr>
<tr>
<td>mlr_learners_regr.featureless</td>
<td>33</td>
</tr>
<tr>
<td>mlr_learners_regr.rpart</td>
<td>34</td>
</tr>
<tr>
<td>mlr_measures</td>
<td>35</td>
</tr>
<tr>
<td>mlr_measures_classif.acc</td>
<td>36</td>
</tr>
<tr>
<td>mlr_measures_classif.auc</td>
<td>37</td>
</tr>
<tr>
<td>mlr_measures_classif.bacc</td>
<td>38</td>
</tr>
<tr>
<td>mlr_measures_classif.ce</td>
<td>39</td>
</tr>
<tr>
<td>mlr_measures_classif.costs</td>
<td>40</td>
</tr>
<tr>
<td>mlr_measures_classif.dor</td>
<td>42</td>
</tr>
<tr>
<td>mlr_measures_classif.fbeta</td>
<td>43</td>
</tr>
<tr>
<td>mlr_measures_classif.fdr</td>
<td>44</td>
</tr>
<tr>
<td>mlr_measures_classif.fn</td>
<td>45</td>
</tr>
<tr>
<td>mlr_measures_classif.fmr</td>
<td>46</td>
</tr>
<tr>
<td>mlr_measures_classif.fp</td>
<td>48</td>
</tr>
<tr>
<td>mlr_measures_classif.fpr</td>
<td>49</td>
</tr>
<tr>
<td>mlr_measures_classif.logloss</td>
<td>50</td>
</tr>
<tr>
<td>mlr_measures_classif.mcc</td>
<td>51</td>
</tr>
<tr>
<td>mlr_measures_classif.npv</td>
<td>52</td>
</tr>
<tr>
<td>mlr_measures_classif.ppv</td>
<td>54</td>
</tr>
<tr>
<td>mlr_measures_classif.precision</td>
<td>55</td>
</tr>
<tr>
<td>mlr_measures_classif.recall</td>
<td>56</td>
</tr>
<tr>
<td>mlr_measures_classif.sensitivity</td>
<td>57</td>
</tr>
<tr>
<td>mlr_measures_classif.specificity</td>
<td>58</td>
</tr>
<tr>
<td>mlr_measures_classif.tn</td>
<td>60</td>
</tr>
<tr>
<td>mlr_measures_classif.tnr</td>
<td>61</td>
</tr>
<tr>
<td>mlr_measures_classif.tp</td>
<td>62</td>
</tr>
<tr>
<td>mlr_measures_classif.tpr</td>
<td>63</td>
</tr>
<tr>
<td>mlr_measures_debug</td>
<td>64</td>
</tr>
<tr>
<td>mlr_measures_elapsed_time</td>
<td>66</td>
</tr>
<tr>
<td>mlr_measures_oob_error</td>
<td>67</td>
</tr>
<tr>
<td>mlr_measures_regr.bias</td>
<td>68</td>
</tr>
<tr>
<td>mlr_measures_regr.ktau</td>
<td>68</td>
</tr>
<tr>
<td>mlr_measures_regr.ktu</td>
<td>69</td>
</tr>
<tr>
<td>mlr_measures_regr.mae</td>
<td>70</td>
</tr>
<tr>
<td>mlr_measures_regr.mape</td>
<td>71</td>
</tr>
<tr>
<td>mlr_measures_regr.maxae</td>
<td>72</td>
</tr>
<tr>
<td>mlr_measures_regr.medae</td>
<td>73</td>
</tr>
<tr>
<td>mlr_measures_regr.medse</td>
<td>74</td>
</tr>
<tr>
<td>mlr_measures_regr.mse</td>
<td>75</td>
</tr>
</tbody>
</table>
R topics documented:

- mlr_measures_regr.msle ............................. 76
- mlr_measures_regr.pbias ........................... 77
- mlr_measures_regr.rae ............................. 78
- mlr_measures_regr.rmse ............................ 79
- mlr_measures_regr.rmsle ............................ 80
- mlr_measures_regr.rrse ............................. 81
- mlr_measures_regr.rse .............................. 82
- mlr_measures_regr.rsq .............................. 83
- mlr_measures_regr.sae .............................. 84
- mlr_measures_regr.smape ............................ 85
- mlr_measures_regr.srho ............................ 86
- mlr_measures_regr.sse .............................. 87
- mlr_measures_regr_selected_features ................. 88
- mlr_resamplings .................................. 89
- mlr_resamplings_bootstrap .......................... 90
- mlr_resamplings_custom ............................. 91
- mlr_resamplings_cv ................................ 92
- mlr_resamplings_holdout ............................. 93
- mlr_resamplings_repeated_cv ......................... 94
- mlr_resamplings_subsampling ......................... 96
- mlr_sugar .......................................... 97
- mlr_tasks .......................................... 98
- mlr_tasks_boston_housing ......................... 99
- mlr_tasks_german_credit ............................ 100
- mlr_tasks_iris ..................................... 101
- mlr_tasks_mtcars .................................. 101
- mlr_tasks_pima ................................... 102
- mlr_tasks_sonar ................................... 102
- mlr_tasks_spam ................................... 103
- mlr_tasks_wine ................................... 104
- mlr_tasks_zoo ................................... 104
- mlr_task_generators ................................. 105
- mlr_task_generators_2dnormals ................. 106
- mlr_task_generators_friedman1 ......... 106
- mlr_task_generators_smiley .................. 107
- mlr_task_generators_xor .................. 108
- predict.Learner .................................. 108
- Prediction ......................................... 109
- PredictionClassif ................................ 111
- PredictionRegr .................................. 113
- resample ......................................... 114
- ResampleResult .................................. 115
- Resampling ....................................... 117
- Task .............................................. 120
- TaskClassif ....................................... 125
- TaskGenerator .................................... 127
- TaskRegr .......................................... 128

Index 130
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.

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

as_benchmark_result(x, ...)

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

Arguments

x :: any
    Object to dispatch on, e.g. a ResampleResult.

... :: any
    Currently not used.

Value

(BenchmarkResult).

as_data_backend  Create a Data Backend

Description

Wraps a DataBackend around data.

Usage

as_data_backend(data, ...)

Arguments

data :: any
    Data to create a DataBackend for. For a data.frame() (this includes tibble() from tibble and data.table::data.table()) this function creates a DataBackendDataTable. See methods("as_data_backend") for possible input formats. Note that third-party packages may extend this functionality.

... :: any
    Additional arguments passed to the respective DataBackend method.
as_task.character

Value

DataBackend.

See Also

Other DataBackend: DataBackendDataTable, DataBackendMatrix, DataBackend

Examples

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

---

as_task.character  Object Coercion

Description

S3 generics and methods to coerce to (lists of) Task, Learner, Resampling, and Measure.

Usage

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

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

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

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

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

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

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

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

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)

## 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)

## 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)

## 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 '

## S3 method for class 'NULL'

## S3 method for class 'NULL'

## 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 mlr3misc::Dictionary.

**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

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

---

**benchmark**

Benchmarks 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**

```r
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().

Logging

The mlr3 uses the lgr package for logging. lgr supports multiple log levels which can be queried with getOption("lgr.log_levels").
To suppress output and reduce verbosity, you can lower the log from the default level "info" to "warn":

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

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

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

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

Note

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

Examples

# benchmarking with benchmark_grid()
tasks = lapply(c("iris", "sonar"), tsk)
learners = lapply(c("classif.featureless", "classif.rpart"), lrn)
resamplings = rsm(p("cv").folds = 3)
BenchmarkResult

```

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(
  task = tasks,
  learner = learners,
  resampling = resamplings
)

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

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

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

BenchmarkResult

Container for Results of benchmark()

Description

This is the result container object returned by benchmark(). A BenchmarkResult consists of the data row-binded data of multiple ResampleResults, which can easily be re-constructed.
Note that all stored objects are accessed by reference. Do not modify any object without cloning it first.

**Format**

`R6::R6Class` object.

**Construction**

```r
bmr = BenchmarkResult$new(data = data.table())
```

- `data::data.table::data.table()`
  Table with data for one resampling iteration per row: `Task`, `Learner`, `Resampling`, iteration `integer(1)`, `Prediction`, and the unique hash `uhash` `character(1)` of the corresponding `ResampleResult`. Additional columns are kept in the resulting object.

**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. Package develops on the other hand may opt to add additional columns here. These columns are preserved in all mutators.

- `rr_data::data.table::data.table()`
  Internal data storage with one row per ResampleResult. Can be joined with $data by joining on column "hash". Not used in mlr3 directly, but can be exploited by add-on packages. Package develops may opt to add additional columns here. These columns are preserved in all mutators.

- `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".

- `tasks::data.table::data.table()`
  Table of used tasks with three columns: "task_hash" `character(1)`, "task_id" `character(1)` and "task" (Task).

- `learners::data.table::data.table()`
  Table of used learners with three columns: "learner_hash" `character(1)` ,"learner_id" `character(1)` and "learner" (Learner). Note that it is not feasible to access learnt models via this getter, as the training task would be ambiguous. Instead, select a row from the table returned by $score().

- `resamplings::data.table::data.table()`
  Table of used resamplings with three columns: "resampling_hash" `character(1)`, "resampling_id" `character(1)` and "resampling" (Resampling).

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

- `uhashes::character()`
  Vector of unique hashes of all included ResampleResults.
BenchmarkResult

Methods

• aggregate(measures = NULL, ids = TRUE, uhashes = FALSE, params = FALSE, conditions = FALSE)
  (list of Measure, logical(1), logical(1), logical(1), logical(1)) -> data.table::data.table()
  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, the following parameters can be set to extract more information from the
  returned ResampleResult:
  – uhashes :: logical(1)
    Adds the uhash values of the ResampleResult as extra character column "uhash".
  – ids :: logical(1)
    Adds object ids ("task_id", "learner_id", "resampling_id") as extra character columns.
  – 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 col-
    umn "warnings", and the number of resampling iterations with errors as extra integer
    column "errors".

• score(measures = NULL, ids = TRUE)
  (list of Measure, logical(1)) -> data.table::data.table()
  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 per-
  formance measures and binds the performance as extra columns. These columns are named
  using the id of the respective Measure.

• resample_result(i = NULL, uhash = NULL)
  (integer(1), character(1)) -> ResampleResult
  Retrieve the i-th ResampleResult, by position or by unique hash uhash. i and uhash are
  mutually exclusive.

• combine(bmr)
  (BenchmarkResult | NULL) -> self
  Fuses a second BenchmarkResult into itself, mutating the BenchmarkResult in-place. If bmr
  is NULL, simply returns self.

• filter(task_ids = NULL, learner_ids = NULL, resampling_ids = NULL)
  (character(), character(), character()) -> self
  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.

• help()
  () -> NULL
  Opens the help page for this object.

S3 Methods

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

Examples

```r
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 = rsm("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

(data.table::data.table()) with the cross product of the input vectors.

Examples

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.

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

Format

R6::R6Class object.

Construction

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

DataBackend$new(data, primary_key = NULL, data_formats = "data.table")
• 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()` constructed with the Matrix package.

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

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

Fields

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

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

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

• rownames :: (integer() | character())
   Returns vector of all distinct row identifiers, i.e. the primary key column.

• hash :: character(1)
   Returns a unique hash for this backend. This hash is cached.

• data_formats :: character()
   Vector of supported data formats. A specific format can be chosen in the $data() method.

Methods

• data(rows = NULL, cols = NULL, format = "data.table")
   (integer() | character(), character()) -> any
   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.

• distinct(rows, cols, na.rm = TRUE)
   (integer() | character(), character(), logical(1)) -> named list()
   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.

   If rows is NULL, all possible distinct values will be returned, even if the value is not present in the data. This affects factor-like variables with empty levels, if supported by the backend.

• head(n = 6)
   integer(1) -> data.table::data.table()
   Returns the first up-to n rows of the data as data.table::data.table().
• \texttt{missings(rows, cols)}
  \begin{verbatim}
  (integer() | character(), character()) \rightarrow named integer()
  \end{verbatim}
  Returns the number of missing values per column in the specified slice of data. Non-existing rows and columns are silently ignored.

\section*{See Also}

Extension Packages: \texttt{mlr3db}

Other DataBackend: \texttt{DataBackendDataTable}, \texttt{DataBackendMatrix}, \texttt{as_data_backend()}

\section*{Examples}

\begin{verbatim}
data = \texttt{data.table::data.table(id = 1:5, x = runif(5), y = sample(letters[1:3], 5, replace = TRUE))}

b = \texttt{DataBackendDataTable$new(data, primary_key = \texttt{"id"})}
print(b)
b$head(2)
b$data(rows = 1:2, cols = \texttt{"x"})
b$distinct(rows = b$rownames, \texttt{"y"})
b$missings(rows = b$rownames, cols = \texttt{names(data)})
\end{verbatim}
Methods

See DataBackend.

See Also

Other DataBackend: DataBackendMatrix, DataBackend, as_data_backend()

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$ncol
b$colnames

# alternative construction
as_data_backend(iris)

Description

DataBackend for Matrix. Data is stored as (sparse) matrix.

Format

R6::R6Class object inheriting from DataBackend.

Construction

DataBackendMatrix$new(data, primary_key = NULL)
as_data_backend(data, primary_key = NULL, ...)

• data :: Matrix::Matrix().
• primary_key :: character(1)
  Not supported by this backend. Rows are addresses by their rownames(). If the matrix does not have row names, integer row indices are used.

Fields

See DataBackend.
**Methods**

See DataBackend.

**See Also**

Other DataBackend: DataBackendDataTable, DataBackend, as_data_backend()

**Examples**

```r
requireNamespace("Matrix")
data = Matrix::Matrix(sample(0:1, 20, replace = TRUE), ncol = 2)
colnames(data) = c("x1", "x2")
rownames(data) = paste0("row", 1:10)

b = as_data_backend(data)
b$head()
b$data(b$rownames[1:3], b$colnames, data_format = "Matrix")
```

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 consist of the following parts:

- Methods train() and predict() which call train_internal() and predict_internal().
- The fitted model, after calling train().
- 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.

Predefined learners are stored in the mlr3misc::Dictionary mlr_learners, e.g. classif.rpart or regr.rpart. A guide on how to extend mlr3 with custom learners can be found in the mlr3book.

Format

R6::R6Class object.

Construction

Note: This object is typically constructed via a derived classes, e.g. LearnerClassif or LearnerRegr.

l = Learner$new(id, task_type, param_set = ParamSet$new(), predict_types = character(), feature_types = character(), properties = character(), packages = character())

- id :: character(1)
  Identifier for the learner.
- task_type :: character(1)
  Type of the task the learner can operator on. E.g., "classif" or "regr".
- param_set :: paradox::ParamSet
  Set of hyperparameters.
- predict_types :: character()
  Supported predict types. Must be a subset of mlr_reflections$learner_predict_types.
- predict_sets :: character()
  Sets to predict on during resample()/benchmark(). Creates and stores a separate Prediction object for each set. The individual sets can be combined via getters in ResampleResult/BenchmarkResult, or Measures can be set to operate on subsets of the calculated prediction sets. Must be a non-empty subset of ("train", "test"). Default is "test".
- feature_types :: character()
  Feature types the learner operates on. Must be a subset of mlr_reflections$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:
Learner

- "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).
- "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).
- "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).

**Fields**

- **id**: character(1)
  Identifier of the learner.

- **task_type**: character(1)
  Stores the type of class this learner can operate on, e.g. "classif" or "regr". A complete list of task types is stored in mlr_reflections$task_types.

- **param_set**: paradox::ParamSet
  Description of available hyperparameters and hyperparameter settings.

- **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.

- **predict_type**: character(1)
  Stores the currently selected predict type. Must be an element of l$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.

- **packages**: character()
  Stores the names of required packages.

- **state**: NULL | named list()
  Current (internal) state of the learner. Contains all information learnt 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.
• encapsulate (named character())
  How to call the code in train_internal() and predict_internal(). Must be a named character vector with names "train" and "predict". Possible values are "none", "evaluate" and "callr". See mlr3misc::encapsulate() for more details.

• 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.

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

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

• timings :: 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, error) and "msg" (character()).

• warnings :: character()
  Returns the logged warnings as vector.

• errors :: character()
  Returns the logged errors as vector.

Methods

• train(task, row_ids = NULL))
  (Task, integer() | character()) -> self
  Train the learner on the row ids of the provided Task. Mutates the learner by reference, i.e. stores the model alongside other objects in field $state.

• predict(task, row_ids = NULL)
  (Task, integer() | character()) -> Prediction
  Uses the data stored during $train() in $state to create a new Prediction based on the provided row_ids of the task.

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

• reset()
  () -> self
  Reset the learner, i.e. un-train by resetting the state.

• help()
  () -> NULL
  Opens the corresponding help page referenced by $man.
Optional Extractors

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

For the following operations, extractors are standardized:

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

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

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

Setting Hyperparameters

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

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

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

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

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

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

See Also

Other Learner: `LearnerClassif`, `LearnerRegr`, `mlr_learners`
LearnerClassif

Description

This Learner specializes Learner for classification problems.

Many predefined learners can be found in the mlr3misc::Dictionary mlr_learners after loading the mlr3learners package.

Format

R6::R6Class object inheriting from Learner.

Construction

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

For a description of the arguments, see Learner. task_type is set to "classif".

Possible values for predict_types are passed to and converted by PredictionClassif:

• "response": Predicts a class label for each observation in the test set.
• "prob": Predicts the posterior probability for each class for each observation in the test set.

Additional learner properties include:

• "twoclass": The learner works on binary classification problems.
• "multiclass": The learner works on multiclass classification problems.

Fields

See Learner.

Methods

See Learner.

See Also

Example classification learners: classif.rpart
Other Learner: LearnerRegr, Learner, mlr_learners
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:
ln = lrn("classif.rpart")
print(ln)

# train the learner:
task = tsk("iris")
ln$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.

Many predefined learners can be found in the `mlr3misc::Dictionary mlr_learners` after loading the `mlr3learners` package.

**Format**

`R6::R6Class` object inheriting from Learner.

**Construction**

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

For a description of the arguments, see Learner. `task_type` is set to "regr".

Possible values for `predict_types` are passed to and converted by `PredictionRegr`:

- "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.

**Fields**

See Learner.

**Methods**

See Learner.
Measure

See Also

Example regression learners: regr.rpart
Other Learner: LearnerClassif, Learner, 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")

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

Description

This is the abstract base class for measures like MeasureClassif and MeasureRegr.
Measures are classes around tailored around two functions:

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

In addition to these two functions, meta-information about the performance measure is stored.
Predefined measures are stored in the mlr3misc::Dictionary mlr_measures, e.g. classif.auc or time_train. A guide on how to extend mlr3 with custom measures can be found in the mlr3book.

Format

R6::R6Class object.

Construction

Note: This object is typically constructed via a derived classes, e.g. MeasureClassif or MeasureRegr.

m = 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_)

• id :: character(1)
  Identifier for the measure.
• task_type :: character(1)
  Type of the task the measure can operator on. E.g., "classif" or "regr".
Measure

- **range :: numeric(2)**
  Feasible range for this measure as \( c(\text{lower\_bound},\text{upper\_bound}) \). 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 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).

- **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 c("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. Note that these packages will be loaded via requireNamespace(), and are not attached.

- **man :: character(1)**
  String in the format [pkg]:[topic] pointing to a manual page for this object.

### Fields

All variables passed to the constructor.
MeasureClassif

Methods

• aggregate(rr)

   ResampleResult -> numeric(1)

   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.

• score(prediction, task = NULL, learner = NULL, train_set = NULL)

   ((named list of) Prediction, Task, Learner, integer() | character()) -> numeric(1)

   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().

• help()

   () -> NULL

   Opens the corresponding help page referenced by $man.

See Also

Other Measure: MeasureClassif, MeasureRegr, mlr_measures

---

MeasureClassif  
Classification Measure

Description

This measure specializes Measure for classification problems. Predefined measures can be found in the mlr3misc::Dictionary mlr_measures.

Format

R6::R6Class object inheriting from Measure.

Construction

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

For a description of the arguments, see Measure. The task_type is set to "classif". Possible values for predict_type are "response" and "prob".

Fields

See Measure.

Methods

See Measure.
See Also

Example classification measures: classif.ce
Other Measure: MeasureRegr, Measure, mlr_measures

MeasureRegr  Regression Measure

Description

This measure specializes Measure for regression problems. Predefined measures can be found in the mlr3misc::Dictionary mlr_measures.

Format

R6::R6Class object inheriting from Measure.

Construction

m = MeasureRegr$new(id, range, minimize = NA, average = "macro", aggregator = NULL, properties = character(), predict_type = "response", predict_sets = "test", task_properties = character(), packages = character())

For a description of the arguments, see Measure. The task_type is set to "regr". Possible values for predict_type are "response" and "se".

Fields

See Measure.

Methods

See Measure.

See Also

Example regression measures: regr.mse
Other Measure: MeasureClassif, Measure, mlr_measures
mlr_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.

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

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

Example learners: `classif.rpart`, `regr.rpart`, `classif.featureless`, `regr.featureless`, `classif.debug`

Sugar function: `lrn()`

Extension Packages: `mlr3learners`

Other Dictionary: `mlr_measures`, `mlr_resamplings`, `mlr_task_generators`, `mlr_tasks`

Other Learner: `LearnerClassif`, `LearnerRegr`, `Learner`

Examples

```r
as.data.table(mlr_learners)
mlr_learners$get("classif.featureless")
lrn("classif.rpart")
```
**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**: Outputs a message during train if the parameter value exceeds \( \text{runif}(1) \).
- **message_predict**: Outputs a message during predict if the parameter value exceeds \( \text{runif}(1) \).
- **warning_train**: Signals a warning during train if the parameter value exceeds \( \text{runif}(1) \).
- **warning_predict**: Signals a warning during predict if the parameter value exceeds \( \text{runif}(1) \).
- **error_train**: Raises an exception during train if the parameter value exceeds \( \text{runif}(1) \).
- **error_predict**: Raises an exception during predict if the parameter value exceeds \( \text{runif}(1) \).
- **segfault_train**: Provokes a segfault during train if the parameter value exceeds \( \text{runif}(1) \).
- **segfault_predict**: Provokes a segfault during predict if the parameter value exceeds \( \text{runif}(1) \).
- **predict_missing**: Ratio of predictions which will be NA.
- **save_tasks**: Saves input task in `model` slot during training and prediction.
- **x**: Numeric parameter. Has no effect.

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

**Format**

R6::R6Class inheriting from LearnerClassif.

**Construction**

```r
LearnerClassifDebug$new()
mlr_learners$get("classif.debug")
lrn("classif.debug")
```

**See Also**

Dictionary of Learners: mlr_learners

as.data.table(mlr_learners) for a complete table of all (also dynamically created) Learner implementations.
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. Hyperparameter method determines the mode of operation during prediction:

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

**Format**

R6::R6Class inheriting from LearnerClassif.

**Construction**

LearnerClassifFeatureless$new()
mlr_learners$get("classif.featureless")
lrn("classif.featureless")

**See Also**

Dictionary of Learners: mlr_learners

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

**Format**

R6::R6Class inheriting from LearnerClassif.

**Construction**

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

**References**


**See Also**

Dictionary of Learners: `mlr_learners`

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

---

**mlr_learners_regr.featureless**

*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 \( \text{mean}(y) \) as response and \( \text{sd}(y) \) as standard error. If `robust` is TRUE, \( \text{median}() \) and \( \text{madn}() \) are used instead of \( \text{mean}() \) and \( \text{sd}() \), respectively.

**Format**

R6::R6Class inheriting from LearnerRegr.
mlr_learners_regr.rpart

Construction

LearnerRegrFeatureless$new()
mlr_learners$get("regr.featureless")
lrn("regr.featureless")

See Also

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

mlr_learners_regr.rpart

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.

Format

R6::R6Class inheriting from LearnerRegr.

Construction

LearnerRegrRpart$new()
mlr_learners$get("regr.rpart")
lrn("regr.rpart")

References


See Also

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

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

Example measures: classif.auc, time_train Sugar function: msr()

Other Dictionary: mlr_learners, mlr_resamplings, mlr_task_generators, mlr_tasks

Other Measure: MeasureClassif, MeasureRegr, Measure

Examples

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

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

Format

`R6::R6Class()` inheriting from `Measure`.

Construction

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.acc")
msr("classif.acc")
```

Meta Information

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

Note

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

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

See Also

Dictionary of Measures: `mlr_measures`

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

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

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.

Format

R6::R6Class() inheriting from Measure.

Construction

This measure can be retrieved from the dictionary mlr_measures:

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

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

**Format**

*R6::R6Class()* inheriting from *Measure*.

**Construction**

This measure can be retrieved from the dictionary *mlr_measures*:

```r
mlr_measures$get("classif.bacc")
msr("classif.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.costs**

**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.

**Format**

`R6::R6Class()` inheriting from `MeasureClassif`.

**Construction**

`MeasureClassifCosts$new(costs = NULL, normalize = TRUE)`

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

`msr("classif.costs")`
mlr_measures_classif.costs

• costs :: matrix()
  Numeric matrix of costs (truth in columns, predicted response in rows).
• normalize :: logical(1)
  If TRUE, calculate the mean cost per observation instead of the total costs.

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

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.bacc, mlr_measures_classif.ce, mlr_measures_classif.logloss

Examples

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

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

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

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

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

Description

Binary classification measure defined as

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

Format

R6::R6Class() inheriting from Measure.

Construction

This measures can be retrieved from the dictionary mlr_measures:

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

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.

Format
`R6::R6Class()` inheriting from `Measure`.

Construction
This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.fbeta")
msr("classif.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, \( \text{NaN} \) is returned. This can be customized by setting the field `na_value`.

mlr_measures_classif.fdr

*False Discovery Rate*

**Description**

Binary classification measure defined as

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

**Format**

R6::R6Class() inheriting from Measure.

**Construction**

This measures can be retrieved from the dictionary mlr_measures:

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

**Meta Information**

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

See Also

Dictionary of Measures: mlr_measures

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


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

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


---

**mlr_measures_classif.fn**

*False Negatives*

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

**Format**

`R6::R6Class()` inheriting from `Measure`.

**Construction**
This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.fn")
msr("classif.fn")
```
Meta Information

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

Note

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

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

See Also

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


--

\texttt{mlr\_measures\_classif\_fnr}

\textit{False Negative Rate}

Description

Binary classification measure defined as 

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

Also known as "miss rate".
Format

R6::R6Class() inheriting from Measure.

Construction

This measures can be retrieved from the dictionary mlr_measures:

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


### mlr_measures_classif.fomr

*False Omission Rate*

#### Description

Binary classification measure defined as

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

#### Format

`R6::R6Class()` inheriting from `Measure`.

#### Construction

This measure can be retrieved from the dictionary `mlr_measures`:

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

**False Positives**

**Description**

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

**Format**

`R6::R6Class()` inheriting from `Measure`.

**Construction**

This measures can be retrieved from the dictionary `mlr_measures`:

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

**Meta Information**

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

**Note**

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

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

Dictionary of Measures: mlr_measures

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


---

mlr_measures_classif.fpr

*False Positive Rate*

Description

Binary classification measure defined as

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

Also known as false out or probability of false alarm.

Format

*R6::R6Class()* inheriting from *Measure*.

Construction

This measure can be retrieved from the dictionary *mlr_measures*:

```r
mlr_measures$get("classif.fpr")
msr("classif.fpr")
```

Meta Information

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

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

**See Also**

Dictionary of Measures: `mlr_measures`

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


---

**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 \(i\).

**Format**

`R6::R6Class()` inheriting from `Measure`.

**Construction**

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.logloss")
msr("classif.logloss")
```
Meta Information

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

Note

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

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

See Also

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


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

\begin{equation}
\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}
\end{equation}
\begin{equation}
\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}
\end{equation}

\textbf{Description}

Binary classification measure defined as

\begin{equation}
\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}
\end{equation}
\begin{equation}
\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}
\end{equation}

\textbf{Format}

\texttt{R6::R6Class()} inheriting from \texttt{Measure}.  

\textit{Matthews Correlation Coefficient}
**mlr_measures_classif.mcc**

**Construction**

This measure can be retrieved from the dictionary `mlr_measures`:

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

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

Format

R6::R6Class() inheriting from Measure.

Construction

This measures can be retrieved from the dictionary mlr_measures:

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


Description

Binary classification measure defined as

\[
\text{Positive Predictive Value} = \frac{TP}{TP + FP}
\]

Also know as "precision".

Format

R6::R6Class() inheriting from Measure.

Construction

This measures can be retrieved from the dictionary mlr_measures:

mlr_measures$get("classif.ppv")
msr("classif.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.
**mlr_measures_classif.precision**

**Positive Predictive Value**

**Description**

Binary classification measure defined as

\[
\text{TP} / (\text{TP} + \text{FP})
\]

Also known as "precision".

**Format**

*R6::R6Class()* inheriting from Measure.

**Construction**

This measure can be retrieved from the dictionary *mlr_measures*:

```r
mlr_measures$get("classif.precision")
msr("classif.precision")
```

**Meta Information**

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

**See Also**

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


Note

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

See Also

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


---

**mlr_measures_classif.recall**

*True Positive Rate*

**Description**

Binary classification measure defined as

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

Also known as "recall" or "sensitivity".

**Format**

`R6::R6Class()` inheriting from `Measure`.

**Construction**

This measure can be retrieved from the dictionary `mlr_measures`:

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

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

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

R6::R6Class() inheriting from Measure.

Construction

This measures can be retrieved from the dictionary mlr_measures:

```r
classif.sensitivity
```

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

Meta Information

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

Note

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

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

See Also

Dictionary of Measures: mlr_measures

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

Other classification measures: mlr_measures_classif.acc, mlr_measures_classif.auc, mlr_measures_classif.bacc,
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.mcc,
mlr_measures_classif.npv, mlr_measures_classif.ppv, mlr_measures_classif.precision,
mlr_measures_classif.recall, mlr_measures_classif.specificity, mlr_measures_classif.tnr,
mlr_measures_classif.tn, mlr_measures_classif.tpr, mlr_measures_classif.tp

Other binary classification measures: mlr_measures_classif.auc, mlr_measures_classif.dor,
mlr_measures_classif.fbeta, mlr_measures_classif.fdr, mlr_measures_classif.fnr, mlr_measures_classif.fomr,
mlr_measures_classif.fpr, mlr_measures_classif.fp, mlr_measures_classif.mcc,
mlr_measures_classif.npv, mlr_measures_classif.ppv, mlr_measures_classif.precision,
mlr_measures_classif.recall, mlr_measures_classif.specificity, mlr_measures_classif.tnr,
mlr_measures_classif.tn, mlr_measures_classif.tpr, mlr_measures_classif.tp
**mlr_measures_classif.specificity**

*True Negative Rate*

**Description**

Binary classification measure defined as

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

Also known as "specificity".

**Format**

R6::R6Class() inheriting from Measure.

**Construction**

This measure can be retrieved from the dictionary `mlr_measures`:

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

**mlr_measures_classif.tn**

*True Negatives*

**Description**

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

**Format**

`R6::R6Class()` inheriting from `Measure`.

**Construction**

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$<("classif.tn")
msr("classif.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`. 
**mlr_measures_classif.tnr**

**True Negative Rate**

**Description**

Binary classification measure defined as

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

Also known as "specificity".

**Format**

R6::R6Class() inheriting from Measure.

**Construction**

This measure can be retrieved from the dictionary mlr_measures:

```r
mlr_measures$get("classif.tnr")
msr("classif.tnr")
```

**Meta Information**

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

See Also

Dictionary of Measures: mlr_measures

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


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.

**Format**

`R6::R6Class()` inheriting from `Measure`.

**Construction**

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.tp")
msr("classif.tp")
```
**Meta Information**

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

**Note**

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

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

**See Also**

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


Format

\texttt{R6::R6Class()} inheriting from \texttt{Measure}.

Construction

This measure can be retrieved from the dictionary \texttt{mlr_measures}:

\begin{verbatim}
mlr_measures$get("classif.tpr")
msr("classif.tpr")
\end{verbatim}

Meta Information

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

Note

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

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

See Also

Dictionary of Measures: \texttt{mlr_measures}

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


mlr_measures_debug

Debug Measure

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

Format
R6::R6Class() inheriting from Measure.

Construction
MeasureDebug$new(na_ratio = 0)
mlr_measures$get("debug")
msr("debug")

- na_ratio::numeric(1)
  Ratio of scores which should be NA. Default is 0.

Fields
- na_ratio::numeric(1).

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

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

Examples

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

*Elapsed Time Measure*

**Description**

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

**Format**

R6::R6Class() inheriting from Measure.

**Construction**

MeasureElapsedTime$new(id, stages)

mlr_measures$get("time_train")
mlr_measures$get("time_predict")
mlr_measures$get("time_both")

msr$get("time_train")
msr$get("time_predict")
msr$get("time_both")

- **id** :: character(1)
  
  Id for the created measure.

- **stages** :: character()
  
  Subset of ("train", "predict"). The runtime of all stages will be summed.

**Meta Information**

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

**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.bias**

### Description

Regression measure defined as

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

Good predictions score close to 0.
**mlr_measures_regr.ktau**

**Format**

\texttt{\texttt{R6::R6Class()}} inheriting from \texttt{Measure}.

**Construction**

This measure can be retrieved from the dictionary \texttt{mlr_measures}:

\begin{verbatim}
mlr_measures$get("classif.bias")
msr("classif.bias")
\end{verbatim}

**Meta Information**

- Type: "regr"
- Range: \((-\infty, \infty)\)
- Minimize: \texttt{NA}
- Required prediction: response

**Note**

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

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

**See Also**

Dictionary of Measures: \texttt{mlr_measures}

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

Other regression measures: \texttt{mlr_measures_regr.ktau}, \texttt{mlr_measures_regr.mae}, \texttt{mlr_measures_regr.mape}, \texttt{mlr_measures_regr.maxae}, \texttt{mlr_measures_regr.medae}, \texttt{mlr_measures_regr.medse}, \texttt{mlr_measures_regr.mse}, \texttt{mlr_measures_regr.msle}, \texttt{mlr_measures_regr.pbias}, \texttt{mlr_measures_regr.rae}, \texttt{mlr_measures_regr.rmse}, \texttt{mlr_measures_regr.rmsle}, \texttt{mlr_measures_regr.pbias}, \texttt{mlr_measures_regr.rse}, \texttt{mlr_measures_regr.rsq}, \texttt{mlr_measures_regr.sae}, \texttt{mlr_measures_regr.smape}, \texttt{mlr_measures_regr.srho}, \texttt{mlr_measures_regr.sse}

**Description**

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

**Format**

\texttt{\texttt{R6::R6Class()}} inheriting from \texttt{Measure}. 

\textit{Kendall’s tau}
Construction

This measures can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.ktau")
msr("classif.ktau")
```

Meta Information

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

Note

The score function calls `mlr3measures::ktau()` from package `mlr3measures`. If the measure is undefined for the input, \(\text{NaN}\) 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.mae`  *Mean Absolute Errors*

Description

Regression measure defined as

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

Format

`R6::R6Class()` inheriting from `Measure`.
**mlr_measures_regr.mape**

**Construction**

This measures can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.mae")
msr("classif.mae")
```

**Meta Information**

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

**Note**

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

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

**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.mape**

*Mean Absolute Percent Error*

**Description**

Regression measure defined as

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

**Format**

`R6::R6Class()` inheriting from `Measure`. 
Construction

This measures can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.mape")
msr("classif.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`.

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.maxae**

*Max Absolute Error*

Description

Regression measure defined as

\[
\max(|y_i - \hat{y}_i|).
\]

Format

`R6::R6Class()` inheriting from `Measure`.  

```r
```
Construction

This measure can be retrieved from the dictionary `mlr_measures`:

```r
classif.maxae()
classif.maxae()
```

Meta Information

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

Note

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

See Also

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

---

**mlr_measures_regr.medae**

*Median Absolute Errors*

**Description**

Regression measure defined as

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

**Format**

`R6::R6Class()` inheriting from `Measure`. 
**Construction**

This measure can be retrieved from the dictionary `mlr_measures`:

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

**Meta Information**

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

**Note**

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

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

**See Also**

- Dictionary of Measures: `mlr_measures`

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


---

**mlr_measures_regr.medse**

*Median Squared Error*

**Description**

Regression measure defined as

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

**Format**

`R6::R6Class()` inheriting from `Measure`. 
mlr_measures_regr.mse

Construction

This measure can be retrieved from the dictionary mlr_measures:

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

Meta Information

- **Type**: "regr"
- **Range**: [0, ∞)
- **Minimize**: TRUE
- **Required prediction**: response

Note

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

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

See Also

Dictionary of Measures: mlr_measures

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


---

**mlr_measures_regr.mse**  Mean Squared Error

Description

Regression measure defined as

$$\frac{1}{n} \sum_{i=1}^{n} (t_i - r_i)^2.$$ 

Format

`R6::R6Class()` inheriting from `Measure`.  

---
Construction

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.mse")
msr("classif.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.


---

```r
mlr_measures_regr.msle
```

*M Mean Squared Log Error*

Description

Regression measure defined as

$$\frac{1}{n} \sum_{i=1}^{n} \left(\ln(1 + t_i) - \ln(1 + r_i)\right)^2.$$  

Format

`R6::R6Class()` inheriting from `Measure`.  

---
Construction

This measures can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.msle")
msr("classif.msle")
```

Meta Information

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

Note

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

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

See Also

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

Construction

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.pbias")
msr("classif.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

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

Can be interpreted as absolute error of the predictions relative to a naive model predicting the mean.

Format

`R6::R6Class()` inheriting from `Measure`.
Construction

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.rae")
msr("classif.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

$$\sqrt{\frac{1}{n} \sum_{i=1}^{n} (t_i - r_i)^2}.$$ 

Format

`R6::R6Class()` inheriting from `Measure`. 
**Construction**

This measures can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.rmse")
msr("classif.rmse")
```

**Meta Information**

- **Type:** "regr"
- **Range:** [0, ∞)
- **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

\[
\sqrt{\frac{1}{n} \sum_{i=1}^{n} (\ln(1 + t_i) - \ln(1 + r_i))^2}.
\]

**Format**

`R6::R6Class()` inheriting from `Measure`. 
Construction

This measure can be retrieved from the dictionary \texttt{mlr_measures}:

\begin{verbatim}
mlr_measures$get("classif.rmsle")
msr("classif.rmsle")
\end{verbatim}

Meta Information

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

Note

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

See Also

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

Other regression measures: \texttt{mlr_measures_regr.bias, mlr_measures_regr.ktau, mlr_measures_regr.mae, mlr_measures_regr.mape, mlr_measures_regr.maxae, mlr_measures_regr.medae, mlr_measures_regr.medse, mlr_measures_regr.mse, mlr_measures_regr.msle, mlr_measures_regr.pbias, mlr_measures_regr.rae, mlr_measures_regr.rmse, mlr_measures_regr.rse, mlr_measures_regr.rse, mlr_measures_regr.rsq, mlr_measures_regr.sae, mlr_measures_regr.smape, mlr_measures_regr.srho, mlr_measures_regr.sse}

\begin{verbatim}
mlr_measures_regr.rse
\end{verbatim}

\textit{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.

Format

\texttt{R6::R6Class()} inheriting from \texttt{Measure}. 
Construction

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.rse")
msr("classif.rse")
```

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.

Format

`R6::R6Class()` inheriting from `Measure`.  

**mlr_measures_regr.rsq**

**Construction**

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.rse")
msr("classif.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.

Construction

This measure can be retrieved from the dictionary `mlr_measures`:

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

Meta Information

- **Type**: "regr"
- **Range**: (-∞, 1]
- **Minimize**: FALSE
- **Required prediction**: response

Note

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

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

\[ \sum_{i=1}^{n} |t_i - r_i| . \]

Format

`R6::R6Class()` inheriting from `Measure`. 
Construction

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.sae")
msr("classif.sae")
```

Meta Information

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

Note

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

See Also

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

Construction

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.smape")
msr("classif.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`.

See Also

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


---

`mlr_measures_regr.srho`

*Spearman’s rho*

Description

Regression measures defined as Spearman’s rank correlation coefficient between truth and response.

Calls `stats::cor()` with method set to "spearman".

Format

`R6::R6Class()` inheriting from `Measure`. 
**mlr_measures_regr.sse**  

**Construction**  
This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.srho")
msr("classif.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: `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  

$$
\sum_{i=1}^{n} (t_i - r_i)^2.
$$

**Format**  
`R6::R6Class()` inheriting from `Measure`. 
**mlr_measures_selected_features**

**Construction**

This measure can be retrieved from the dictionary `mlr_measures`:

```r
mlr_measures$get("classif.sse")
msr("classif.sse")
```

**Meta Information**

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

**Note**

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

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

**See Also**

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


---

**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.

**Format**

`R6::R6Class()` inheriting from `Measure`. 

---

**mlr_measures_selected_features**

*Selected Features Measure*
Construction

MeasureSelectedFeatures$new(normalize = FALSE)
mlr_measures$get("selected_features")
msr("selected_features")

- normalize :: logical(1)
  If normalize is set to TRUE, divides the number of features by the total number of features.

Meta Information

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

See Also

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

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().

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", "params", and "iters".
mnl_resamplings_bootstrap

See Also

Example resamplings: cv, bootstrap
Sugar function: rsmp()
Other Dictionary: mnl_learners, mnl_measures, mnl_task_generators, mnl_tasks
Other Resampling: Resampling

Examples

as.data.table(mnl_resamplings)
mnl_resamplings$get("cv")
rsmp("subsampling")

mnl_resamplings_bootstrap

Bootstrap Resampling

Description

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

Format

R6::BaseClass inheriting from Resampling.

Construction

ResamplingBootstrap$new()
mnl_resamplings$get("bootstrap")
rsmp("bootstrap")

Fields

See Resampling.

Methods

See Resampling.

Parameters

- repeats :: integer(1)
  Number of repetitions.
- ratio :: numeric(1)
  Ratio of observations to put into the training set.
See Also

Dictionary of Resamplings: `mlr_resamplings`

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

Examples

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

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

**Format**

*R6::R6Class* inheriting from `Resampling`.

**Construction**

```r
ResamplingCustom$new()
mlr_resamplings$get("custom")
rsmp("custom")
```

**Fields**

See `Resampling`.

**Methods**

See `Resampling`.
See Also

Dictionary of Resamplings: mlr_resamplings
as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) Resampling implementations.

Examples

# 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.

Format

R6::R6Class inheriting from Resampling.

Construction

ResamplingCV$new()
mlr_resamplings$get("cv")
rsmp("cv")

Fields

See Resampling.

Methods

See Resampling.

Parameters

- folds :: integer(1)
  Number of folds.
mlr_resamplings_holdout

See Also

Dictionary of Resamplings: mlr_resamplings
as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) Resampling implementations.

Examples

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

# Instantiate Resampling
rcv = rsmp("cv", folds = 3)
rcv$instantiate(task)

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

# Internal storage:
rcv$instance # table

mlr_resamplings_holdout

Holdout Resampling

Description

Splits data into a training set and a test set. Parameter ratio determines the ratio of observation going into the training set (default: 2/3).

Format

R6::R6Class inheriting from Resampling.

Construction

ResamplingHoldout$new()
mlr_resamplings$get("holdout")
rsmp("holdout")

Fields

See Resampling.

Methods

See Resampling.
Parameters

- ratio:: numeric(1)
  Ratio of observations to put into the training set.

See Also

Dictionary of Resamplings: mlr_resamplings
as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) Resampling implementations.

Examples

```r
# 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
```

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.

Format

*R6::R6Class()* inheriting from Resampling.

Construction

ResamplingRepeatedCV$new()
mlr_resamplings$get("repeated_cv")
rsmp("repeated_cv")
## Fields

See Resampling.

## Methods

See Resampling. Additionally, the class provides two helper function to translate iteration numbers to folds / repeats:

- **folds(iters)**
  - integer() -> integer()
  - Translates iteration numbers to fold number.

- **repeats(iters)**
  - integer() -> integer()
  - Translates iteration numbers to repetition number.

## Parameters

- **repeats** :: integer(1)
  - Number of repetitions.

- **folds** :: integer(1)
  - Number of folds.

## See Also

Dictionary of Resamplings: mlr_resamplings

as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) Resampling implementations.

## Examples

```r
# 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)
interact(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.

Format

R6::R6Class inheriting from Resampling.

Construction

ResamplingSubsampling$new()
mlr_resamplings$get("subsampling")
rmsp("subsampling")

Fields

See Resampling.

Methods

See Resampling.

Parameters

- repeats :: integer(1)
  Number of repetitions.
- ratio :: numeric(1)
  Ratio of observations to put into the training set.

See Also

Dictionary of Resamplings: mlr_resamplings

as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) Resampling implementations.

Examples

# 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`.
- `rsmps()` for a list of Resamplings from `mlr_resamplings`.
- `msr()` for a Measure from `mlr_measures`.
- `msrs()` for a list of Measures from `mlr_measures`.

**Usage**

```r
tsk(.key, ...)

tsk(.key, ...)

tgen().key, ...)

tgens(.keys, ...)

lrn(.key, ...)

lrns(.keys, ...)
	rsmp(.key, ...)
```

---
rsmps(.keys, ...)

msr(.key, ...)

msrs(.keys, ...)

### Arguments

- **.key** :: character(1)
  
  Key passed to the respective mlr3misc::Dictionary to retrieve the object.

- **...** :: named list()
  
  Named arguments passed to the constructor, to be set as parameters in the paradox::ParamSet, or to be set as public field. See mlr3misc::dictionary_sugar_get() for more details.

- **.keys** :: character()
  
  Keys passed to the respective mlr3misc::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
tsk("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")
```

---

### 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.

For a more convenient way to retrieve and construct tasks, see tsk().

### 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

Example tasks: iris (multi-class classification), spam (binary classification), boston_housing (regression)
Sugar function: tsk()
Other Dictionary: mlr_learners, mlr_measures, mlr_resamplings, mlr_task_generators
Other Task: TaskClassif, TaskRegr, TaskSupervised, Task

Examples

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
mlr_tasks_german_credit

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.

mlr_tasks_german_credit

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.

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")

mlr_tasks_mtcars Motor Trend Regression Task

Description

A regression task for the datasets::mtcars data set. Target variable is mpg (Miles/(US) gallon).

Format

R6::R6Class inheriting from TaskRegr.
Construction

```r
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.

---

`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

```r
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.

---

`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`.

Construction

```r
mlr_tasks$get("sonar")
tsk("sonar")
```
See Also

Dictionary of Tasks: mlr_tasks
as.data.table(mlr_tasks) for a complete table of all (also dynamically created) Tasks.

---

**mlr_tasks_spam**  
*Spam Classification Task*

**Description**

Spam data set from the UCI machine learning repository ([http://archive.ics.uci.edu/ml/datasets/spambase](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**

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

**Source**

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

**References**


**See Also**

Dictionary of Tasks: mlr_tasks
as.data.table(mlr_tasks) for a complete table of all (also dynamically created) Tasks.
### mlr_tasks_wine

**Wine Classification Task**

**Description**

Wine data set from the UCI machine learning repository ([https://archive.ics.uci.edu/ml/datasets/wine](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**

```r
mlr_tasks$get("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.

### mlr_tasks_zoo

**Zoo Classification Task**

**Description**

A classification task for the `mlbench::Zoo` data set.

**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.

---

**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()`.

**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**

Example generators: `xor`

Sugar function: `tgen()`

Other Dictionary: `mlr_learners, mlr_measures, mlr_resamplings, mlr_tasks`

Other TaskGenerator: `TaskGenerator`

**Examples**

```r
mlr_task_generators$get("smiley")
tgen("2dnormals")
```
mlr_task_generators_2dnormals

2D Normals Classification Task Generator

Description

A TaskGenerator for the 2d normals task in mlbench::mlbench.2dnormals().

Format

R6::R6Class inheriting from TaskGenerator.

Construction

TaskGenerator2DNormals$new()
mlr_task_generators$get("2dnormals")
tgen("2dnormals")

See Also

Dictionary of TaskGenerators: mlr_task_generators
as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) TaskGenerator implementations.

Examples

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

---

mlr_task_generators_friedman1

Friedman1 Regression Task Generator

Description

A TaskGenerator for the friedman1 task in mlbench::mlbench.friedman1().

Format

R6::R6Class inheriting from TaskGenerator.

Construction

TaskGeneratorFriedman1$new()
mlr_task_generators$get("friedman1")
tgen("friedman1")
See Also

Dictionary of TaskGenerators: mlr_task_generators

as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) TaskGenerator implementations.

Examples

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

XOR Classification Task Generator

Description

A TaskGenerator for the xor task in mlbench::mlbench.xor().

Format

R6::R6Class inheriting from TaskGenerator.

Construction

TaskGeneratorXor$new()
mlr_task_generators$get("xor")
tgen("xor")

See Also

Dictionary of TaskGenerators: mlr_task_generators
as.data.table(mlr_resamplings) for a complete table of all (also dynamically created) TaskGenerator implementations.

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, ...)
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

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.

Format

R6::R6Class object.

Construction

This object is constructed via a derived classes, e.g. PredictionClassif or PredictionRegr.
Prediction

Fields

- **row_ids**: (integer() | character())
  Vector of row ids for which predictions are stored.

- **truth**: any
  True (observed) outcome.

- **task_type**: character(1)
  Stores the type of the Task.

- **task_properties**: character()
  Stores properties of the Task.

- **predict_types**: character()
  Vector of predict types this object stores.

- **missing**: logical()
  Returns row_ids for which the predictions are missing or incomplete.

Methods

- **score(measures = NULL, task = NULL, learner = NULL)**
  (list of Measure, Task, Learner) -> Prediction
  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.

- **help()**
  () -> NULL
  Opens the help page for this object.

S3 Methods

- **as.data.table(rr)**
  Prediction -> data.table::data.table()
  Converts the data to a data.table::data.table().

- **c(...)**,keep_duplicates = TRUE)
  (Prediction, Prediction, ...) -> Prediction
  Combines multiple Predictions to a single Prediction. If keep_duplicates is FALSE and there are duplicated row ids, the data of the former passed objects get overwritten by the data of the later passed objects.

See Also

Other Prediction: PredictionClassif, PredictionRegr
PredictionClassif

**Prediction Classif 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.

**Format**

R6::R6Class object inheriting from Prediction.

**Construction**

p = PredictionClassif$new(task = NULL, row_ids = task$row_ids, truth = task$truth(), response = NULL, prob = NULL)

- **task** :: TaskClassif
  Task, used to extract defaults for row_ids and truth.
- **row_ids** :: (integer() | character())
  Row ids of the observations in 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".

**Fields**

All fields from Prediction, and additionally:

- **response** :: factor()
  Access to the stored predicted class labels.
- **prob** :: matrix()
  Access to the stored probabilities.
- **confusion** :: matrix()
  Confusion matrix resulting from the comparison of truth and response. Truth is in columns, predicted response is in rows.

The field task_type is set to "classif".
Methods

- **set_threshold(th)**
  numeric() -> self
  Sets the prediction response based on the provided threshold. See the section on thresholding for more information.

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

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.

## Format

* `R6::R6Class` object inheriting from `Prediction`.

## Construction

```r
p = PredictionRegr$new(task = NULL, row_ids = task$row_ids, truth = task$truth(), response = NULL, se = NULL)
```

- **task** :: `TaskRegr`  
  Task, used to extract defaults for `row_ids` and `truth`.
- **row_ids** :: `(integer() | character())`  
  Row ids of the observations in 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.

## Fields

All fields from `Prediction`, and additionally:

- **response** :: `numeric()`  
  Access to the stored predicted response.
- **se** :: `numeric()`  
  Access to the stored standard error.

The field `task_type` is set to "regr".

## 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).

Usage

```r
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().

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
g := get_logger("mlr3")$set_threshold("debug")
```

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

**Note**

The fitted models are discarded after the predictions have been scored 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))
```

---

**ResampleResult**

*Container for Results of resample()*

**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.
ResampleResult

Format

R6::R6Class object.

Construction

$rr = \text{ResampleResult}\$\text{\$new(data, uhash = NULL)}$

- $\text{data} :: \text{data.table::data.table()}$
  - Table with data for one resampling iteration per row: Task, Learner, Resampling, iteration (integer(1)), and Prediction.
- $\text{uhash} :: \text{character(1)}$
  - Unique hash for this ResampleResult. If NULL, a new unique hash is generated. This unique hash is primarily needed to group information in BenchmarkResults.

Fields

- $\text{data} :: \text{data.table::data.table()}$
  - Internal data storage. We discourage users to directly work with this field.
- $\text{task} :: \text{Task}$
  - The task resample() operated on.
- $\text{learners} :: \text{list of Learner}$
  - List of trained learners, sorted by resampling iteration.
- $\text{resampling} :: \text{Resampling}$
  - Instantiated Resampling object which stores the splits into training and test.
- $\text{warnings} :: \text{data.table::data.table()}$
  - Returns 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.
- $\text{errors} :: \text{data.table::data.table()}$
  - Returns 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.
- $\text{uhash} :: \text{character(1)}$
  - Unique hash for this object.

Methods

- $\text{predictions(predict_sets = "test")}$
  - character() -> list of Prediction
  - List of prediction objects, sorted by resampling iteration. If multiple sets are given, these are combined to a single one for each iteration.
- $\text{prediction(predict_sets = "test")}$
  - character() -> 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.
• score(measures = NULL, ids = TRUE)
  (list of Measure, logical(1)) -> data.table::data.table()
  Returns a table with one row for each resampling iteration, including all involved objects: Task, Learner, Resampling, iteration number (integer(1)), and Prediction. A column with the individual (per resampling iteration) performance is added for each Measure, named with the id of the respective measure. If ids is TRUE, extra columns with the ids of objects ("task_id", "learner_id", "resampling_id") are binded to the table to allow a more convenient subsetting. If measures is NULL, measures defaults to the return value of default_measures().

• aggregate(measures = NULL)
  list of Measure -> named numeric()
  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().

• help()
  () -> NULL
  Opens the help page for this object.

S3 Methods

• as.data.table(rr)
  ResampleResult -> data.table::data.table()
  Returns a copy of the internal data.

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  Resampling Class

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 mlr3misc::Dictionary mlr_resamplings, e.g. cv or bootstrap.
Resampling

Format

R6::R6Class object.

Construction

Note: This object is typically constructed via a derived classes, e.g. ResamplingCV or ResamplingHoldout.

\[ r = \text{Resampling}\$\text{new}(\text{id}, \text{param}\_\text{set}, \text{duplicated}\_\text{ids} = \text{FALSE}, \text{man} = \text{NA}\_\text{character}_\_) \]

- id :: character(1)
  Identifier for the resampling strategy.
- param_set :: paradox::ParamSet
  Set of hyperparameters.
- duplicated_ids :: logical(1)
  Set to TRUE if this resampling strategy may have duplicated row ids in a single training set or test set.
- man :: character(1)
  String in the format [pkg]:[topic] pointing to a manual page for this object.

Fields

All variables passed to the constructor, and additionally:

- iters :: integer(1)
  Return the number of resampling iterations, depending on the values stored in the param_set.
- instance :: any
  During instantiate(), the instance is stored in this slot. The instance can be in any arbitrary format.
- is\_instantiated :: logical(1)
  Is TRUE, if the resampling has been instantiated.
- 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().
- hash :: character(1)
  Hash (unique identifier) for this object.
  E.g., this is TRUE for Bootstrap, and FALSE for cross validation. Only used internally.

Methods

- instantiate(task)
  Task -> self
  Materializes fixed training and test splits for a given task and stores them in r$instance.
- train_set(i)
  integer(1) -> (integer() \| character())
  Returns the row ids of the i-th training set.
• **test_set(i)**
  integer(1) -> (integer() | character())
  Returns the row ids of the i-th test set.

• **help()**
  () -> NULL
  Opens the corresponding help page referenced by $man.

### Stratification

All derived classes support stratified sampling. The stratification variables are assumed to be discrete and must be stored in the Task with column role "stratum". In case of multiple stratification variables, each combination of the values of the stratification variables forms a strata.

First, the observations are divided into subpopulations based on one or multiple stratification variables (assumed to be discrete), c.f. task$strata.

Second, the sampling is performed in each of the k subpopulations separately. Each subgroup is divided into iter training sets and iter test sets by the derived Resampling. These sets are merged based on their iteration number: all training sets from all subpopulations with iteration 1 are combined, then all training sets with iteration 2, and so on. Same is done for all test sets. The merged sets can be accessed via $train_set(i) and $test_set(i), respectively.

### Grouping / Blocking

All derived classes support grouping of observations. The grouping variable is assumed to be discrete and must be stored in the Task with column role "group".

Observations in the same group are treated like a "block" of observations which must be kept together. These observations either all go together into the training set or together into the test set.

The sampling is performed by the derived Resampling on the grouping variable. Next, the grouping information is replaced with the respective row ids to generate training and test sets. The sets can be accessed via $train_set(i) and $test_set(i), respectively.

### 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

### Examples

```r
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)
```

Task

Description

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 mlr3misc::Dictionary mlr_tasks, e.g. iris or boston_housing.

Format

R6::R6Class object.

Construction

Note: This object is typically constructed via a derived classes, e.g. TaskClassif or TaskRegr.

t = Task$new(id, task_type, backend)

• id :: character(1)
  Identifier for the task.

# 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
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_type :: character(1)`
  Set in the classes which inherit from this class. Must be an element of `mlr_reflections$task_types$type`.

• `backend :: DataBackend`
  Either a `DataBackend`, or any object which is convertible to a DataBackend with `as_data_backend()`.
  E.g., a `data.frame()` will be converted to a `DataBackendDataTable`.

**Fields**

• `backend :: DataBackend`.

• `col_info :: data.table::data.table()`
  Table with with 3 columns:
  - "id" stores the name of the column.
  - "type" holds the storage type of the variable, e.g. integer, numeric or character.
  - "levels" stores a vector of distinct values (levels) for factor variables.

• `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.
  - "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()`, ...).

• `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 a character vector of row ids. To alter the roles, just modify the list, e.g. with R's set functions (`intersect()`, `setdiff()`, `union()`, ...).

• `feature_names :: character()`
  Return all column names with role == "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.
• hash :: character(1)
  Hash (unique identifier) for this object.
• id :: character(1)
  Identifier of the Task.
• ncol :: integer(1)
  Returns the total number of cols with role "target" or "feature".
• nrow :: integer(1)
  Return the total number of rows with role "use".
• row_ids :: (integer() | character())
  Returns the row ids of the DataBackend for observations with with role "use".
• target_names :: character()
  Returns all column names with role "target".
• task_type :: character(1)
  Stores the type of the Task.
• 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 must not be set explicitly.
• strata :: data.table::data.table()
  If the task has columns designated with role "stratum", returns a table with one subpop-
  ulation per row and two columns: N (integer()) with the number of observations in the
  subpopulation and row_id (list of integer() | list of character()) 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() | character()) and the grouping variable group (vector()). Returns NULL if there
  are is no grouping column.
  See Resampling for more information on grouping.
• weights :: data.table::data.table()
  If the task has a column with designated role "weight", table with two columns: row_id
  (integer() | character()) and the observation weights weight (numeric()). Returns NULL
  if there are is no weight column.
• man :: character(1)
  String in the format [pkg]:[topic] pointing to a manual page for this object.

Methods

• data(rows = NULL, cols = NULL, data_format = NULL)
  (integer() | character(), character(), character(1)) -> any
  Returns a slice of the data from the DataBackend in the data format specified by data_format
  (depending on the DataBackend, but usually a data.table::data.table()).
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.

- **formula(rhs = ".")** character() -> stats::formula()
  Constructs a stats::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 ".").

- **levels(cols = NULL)** character() -> named list()
  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 filtering a task, call $droplevels().

- **droplevels(cols = NULL)** character() -> self
  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".

- **missings(cols = NULL)** character() -> named integer()
  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".

- **head(n = 6)**
  integer() -> data.table::data.table()
  Get the first n observations with role "use".

- **set_row_role(rows,new_roles,exclusive = TRUE)**
  (character(), character(), logical(1)) -> self
  Adds the roles new_roles to rows referred to by 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.

- **set_col_role(cols,new_roles,exclusive = TRUE)**
  (character(), character(), logical(1)) -> self
  Adds the roles new_roles to columns referred to by 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.

- **filter(rows)**
  (integer() | character()) -> self
  Subsets the task, reducing it to only keep the rows specified in rows.
  This operation mutates the task in-place. See the section on task mutators for more information.
• **select(cols)**
  character() -> self
  Subsets the task, reducing it to only keep the features specified in 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 informa-
tion.

• **cbind(data)**
  data.frame() -> self
  Adds additional columns to the DataBackend. 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 t$ow_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 informa-
tion.

• **rbind(data)**
  data.frame() -> self
  Adds additional rows to the DataBackend. The new row ids must be provided as column in
data. If this column is missing, new row ids are constructed automatically. In case of name
clashes of row ids, rows in data have higher precedence and virtually overwrite the rows in the
DataBackend.
  This operation mutates the task in-place. See the section on task mutators for more informa-
tion.

• **rename(from,to)**
  (character(), character()) -> self
  Renames columns by mapping column names in old to new column names in new.
  This operation mutates the task in-place. See the section on task mutators for more informa-
tion.

• **help()**
  () -> NULL
  Opens the corresponding help page referenced by $man.

**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 combines 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`.

**See Also**

Other Task: `TaskClassif`, `TaskRegr`, `TaskSupervised`, `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 `mlr3misc::Dictionary mlr_tasks`.

**Format**

`R6::R6Class` object inheriting from `Task/TaskSupervised`.
**Construction**

\[
t = \text{TaskClassif}$\text{new}(\text{id}, \text{backend}, \text{target}, \text{positive} = \text{NULL})
\]

- \text{id} :: character(1)
  Identifier for the task.
- \text{backend} :: DataBackend
  Either a DataBackend, or any object which is convertible to a DataBackend with \text{as\_data\_backend()}. E.g., a data.frame() will be converted to a DataBackendDataTable.
- \text{target} :: character(1)
  Name of the target column.
- \text{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 $\text{class\_names}$ is the positive class, and the second element is the negative class.

**Fields**

All methods from TaskSupervised, and additionally:

- \text{class\_names} :: character()
  Returns all class labels of the target column.
- \text{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.
- \text{negative} :: character(1)
  Stores the negative class for binary classification tasks, and NA for multiclass tasks.

**Methods**

See TaskSupervised.

**See Also**

Example classification tasks: iris
Other Task: TaskRegr, TaskSupervised, Task, mlr_tasks

**Examples**

data("Sonar", package = "mlbench")
task = TaskClassif$new("sonar", backend = Sonar, target = "Class", positive = "M")

task$task$task_type
task$formula()
task$truth()
task$class\_names
task$positive

# possible properties:
mlr_reflections$task\_properties$classif
TaskGenerator

TaskGenerator Class

Description

Creates a Task of arbitrary size. Predefined task generators are stored in the mlr3misc::Dictionary mlr_task_generators, e.g. xor.

Format

R6::R6Class object.

Construction

\begin{verbatim}
g = TaskGenerator$new(id, task_type, packages = character(), param_set = ParamSet$new(), man = NA_character_)
\end{verbatim}

- **id** :: character(1)
  Identifier for the learner.

- **task_type** :: character(1)
  Type of the task the learner can operator on. E.g., "classif" or "regr".

- **packages** :: character()
  Set of required packages. Note that these packages will be loaded via requireNamespace(), and are not attached.

- **param_set** :: paradox::ParamSet
  Set of hyperparameters.

- **man** :: character(1)
  String in the format [pkg]:[topic] pointing to a manual page for this object.

Fields

All variables passed to the constructor, and additionally:

- **task_type** :: character(1)
  Stores the type of class this learner can operate on, e.g. "classif" or "regr". A complete list of task types is stored in mlr_reflections$task_types$type.

Methods

- **generate(n)**
  integer(1) -> Task
  Creates a task of type task_type with n observations, possibly using additional settings stored in param_set.

- **help()**
  () -> NULL
  Opens the corresponding help page referenced by $man.
See Also

Other TaskGenerator: mlr_task_generators

---

TaskRegr

### Description

This task specializes Task and TaskSupervised for regression problems. The target column is assumed to be numeric. The task_type is set to "classif".

Predefined tasks are stored in the mlr3misc::Dictionary mlr_tasks.

### Format

**R6::R6Class** object inheriting from Task/TaskSupervised.

### Construction

\[
t = \text{TaskRegr}\$\text{new}(\text{id}, \text{backend}, \text{target})
\]

- **id**: character(1)
  Identifier for the task.
- **backend**: (DataBackend | data.frame() | ...)
  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.

### Fields

See TaskSupervised.

### Methods

See TaskSupervised.

### See Also

Example regression tasks: boston_housing

Other Task: TaskClassif, TaskSupervised, Task, mlr_tasks
Examples

```r
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, 30
  mlr_measures, 35
  mlr_resamplings, 89
  mlr_task_generators, 105
  mlr_tasks, 98

as_benchmark_result, 6
as_data_backend, 6, 17–19
as_data_backend(), 15
as_learner (as_task.character), 7
as_learners (as_task.character), 7
as_measure (as_task.character), 7
as_measures (as_task.character), 7
as_resampling (as_task.character), 7
as_resamplings (as_task.character), 7
as_task (as_task.character), 7
as_task.character, 7
as_tasks (as_task.character), 7

benchmark, 9
benchmark(), 11, 20, 22, 27, 28, 117
benchmark_grid, 14
benchmark_grid(), 10
BenchmarkResult, 6, 10, 11, 11, 13, 14, 20, 22, 116
bootstrap, 90, 117
boston_housing, 99, 120, 128
classif.auc, 26, 35
classif.ce, 29
classif.debug, 30
classif.featureless, 30
classif.rpart, 20, 24, 30
cv, 90, 117
data.frame(), 10, 109
data.table::copy(), 17
data.table::data.table(), 6, 10, 12–17, 22, 30, 35, 89, 99, 105, 110, 116, 117, 121–124

DataBackend, 6, 7, 15, 17–19, 120–126, 128
DataBackendDataTable, 6, 7, 15–17, 17, 19, 121, 125, 126, 128
DataBackendMatrix, 7, 15–18, 18
datasets::iris, 101
datasets::mtcars, 101
default_measures, 19
default_measures(), 117
expand.grid(), 14
future::plan(), 10, 114
intersect(), 121
iris, 99, 120, 126
LearnerClassif, 20, 23, 24, 26, 30–33, 111
LearnerClassifDebug
  (mlr_learners_classif.debug), 31
LearnerClassifFeatureless
  (mlr_learners_classif.featureless), 32
LearnerClassifRpart
  (mlr_learners_classif.rpart), 33
LearnerRegr
  20, 23, 24, 25, 30, 33, 34, 113
LearnerRegrFeatureless
  (mlr_learners_regr.featureless), 33
LearnerRegrRpart
  (mlr_learners_regr.rpart), 34
Learners, 31–34
lgr::lgr-package, 10, 115
INDEX

llrn (mlr_sugar), 97
llrn(), 30
llrns (mlr_sugar), 97
Matrix::Matrix(), 16, 18
max.col(), 111
mean(), 27
Measure, 7, 13, 19, 20, 26, 28, 29, 35–89, 97,
110, 116, 117
MeasureClassif, 26, 28, 29, 35, 40
MeasureClassifCosts, 100
MeasureClassifCosts
(mlr_measures_classif.costs), 40
MeasureDebug (mlr_measures_debug), 66
MeasureElapsedTime
(mlr_measures_elapsed_time), 67
MeasureOOBError
(mlr_measures_oob_error), 68
MeasureRegr, 26, 28, 29, 35
Measures, 36, 37, 39–42, 44–48, 50–54,
56–60, 62–89
MeasureSelectedFeatures
(mlr_measures_selected_features), 88
mlbench::BostonHousing2, 100
mlbench::mlbench.2dnormals(), 106
mlbench::mlbench.friedman1(), 106
mlbench::mlbench.smiley(), 107
mlbench::mlbench.xor(), 108
mlbench::PimaIndiansDiabetes2, 102
mlbench::Sonar, 102
mlbench::Zoo, 104
mlr3 (mlr3-package), 5
mlr3-package, 5
mlr3measures::acc(), 36
mlr3measures::auc(), 37
mlr3measures::bacc(), 39
mlr3measures::bias(), 69
mlr3measures::ce(), 40
mlr3measures::dor(), 42
mlr3measures::fbeta(), 43
mlr3measures::fdr(), 45
mlr3measures::fn(), 46
mlr3measures::fmr(), 47
mlr3measures::fomr(), 48
mlr3measures::fp(), 49
mlr3measures::fpr(), 51
mlr3measures::ktau(), 70
mlr3measures::logloss(), 52
mlr3measures::mae(), 71
mlr3measures::mape(), 72
mlr3measures::maeae(), 73
mlr3measures::mcc(), 53
mlr3measures::meda(), 74
mlr3measures::medse(), 75
mlr3measures::mse(), 76
mlr3measures::msle(), 77
mlr3measures::nse(), 78
mlr3measures::ppbias(), 79
mlr3measures::npv(), 78
mlr3measures::ppv(), 74
mlr3measures::precision(), 57
mlr3measures::reac(), 79
mlr3measures::recall(), 58
mlr3measures::rmse(), 80
mlr3measures::rmsle(), 81
mlr3measures::rrse(), 82
mlr3measures::rse(), 83
mlr3measures::rsq(), 84
mlr3measures::ssae(), 85
mlr3measures::sensitivity(), 59
mlr3measures::smape(), 86
mlr3measures::specificity(), 60
mlr3measures::srocc(), 87
mlr3measures::sse(), 88
mlr3measures::tn(), 61
mlr3measures::tnr(), 63
mlr3measures::tp(), 64
mlr3measures::tp(), 65
mlr3misc::Dictionary, 9, 20, 24–26, 28–30,
35, 89, 97–99, 103, 117, 120, 125,
127, 128
mlr3misc::dictionary_sugar_get(), 97,
98
mlr3misc::encapsulate(), 22
mlr3misc::insert_named(), 23
mlr3misc::unnest(), 13
mlr_assertions, 9
mlr_coercions (as_task.character), 7
mlr_learners, 20, 23–26, 30, 31–35, 90, 97,
99, 105
mlr_learners_classif.debug, 31
mlr_learners_classif.featureless, 32
mlr_learners_classif.rpart, 33
mlr_learners_regr.featureless, 33
mlr_learners_regr.rpart, 34
mlr_measures, 26, 28–30, 35, 36–90, 97, 99,
mlr_measures_classif.auc, 36, 37, 39–65
mlr_measures_classif.bacc, 36, 37, 38, 40–42, 44–48, 50–54, 56–60, 62–65
mlr_measures_classif.ce, 36, 37, 39, 41, 42, 44–48, 50–54, 56–60, 62–65
mlr_measures_classif.costs, 36, 37, 39, 40, 40, 42, 44–48, 50–54, 56–60, 62–65
mlr_measures_classif.dor, 36–41, 42, 44–63
mlr_measures_classif.fbeta, 36–43, 43, 45–65
mlr_measures_classif.fdr, 36–44, 44, 46–65
mlr_measures_classif.fn, 36–45, 45, 47–63
mlr_measures_classif.fnr, 36–46, 46, 48–63
mlr_measures_classif.fomr, 36–47, 48, 50–65
mlr_measures_classif.fp, 36–49, 49, 51–65
mlr_measures_classif.fpr, 36–50, 50, 52–65
mlr_measures_classif.mcc, 36–52, 52, 54–63
mlr_measures_classif.npv, 36, 38–53, 54, 56–59, 61–65
mlr_measures_classifppv, 36, 38–55, 55, 57–59, 61–65
mlr_measures_classif.precision, 36, 38–56, 56, 58, 59, 61–65
mlr_measures_classif.recall, 36, 38–57, 57, 59, 61–65
mlr_measures_classif.tn, 36, 38–41, 43–47, 49–53, 55–59, 61, 61, 63–65
mlr_measures_classif.tpr, 36, 38–41, 43–47, 49–53, 55–59, 61, 62, 62, 64, 65
mlr_measures_classif.tpr, 36, 38–41, 43–47, 49–53, 55–59, 61–64, 64
mlr_measures_debug, 66
mlr_measures_elapsed_time, 67
mlr_measures_oob_error, 68
mlr_measures_regr.bias, 68, 70–88
mlr_measures_regr.ktau, 69, 69, 71–88
mlr_measures_regr.mae, 69, 70, 70, 72–88
mlr_measures_regr.mapse, 69–71, 71, 73–88
mlr_measures_regr.maxae, 69–72, 72, 74–88
mlr_measures_regr.medae, 69–73, 73, 75–88
mlr_measures_regr.medse, 69–74, 74, 76–88
mlr_measures_regr.mse, 69–75, 75, 77–88
mlr_measures_regr.rsle, 69–76, 76, 78–88
mlr_measures_regr.pbias, 69–77, 77, 79–88
mlr_measures_regr.rae, 69–78, 78, 80–88
mlr_measures_regr.rsle, 69–79, 79, 81–88
mlr_measures_regr.rmsle, 69–80, 80, 82–88
mlr_measures_regr.rse, 69–81, 81, 83–88
mlr_measures_regr.srho, 69–82, 82, 84–88
mlr_measures_regr.smape, 69–83, 83, 85–88
mlr_measures_regr.sae, 69–84, 84, 86–88
mlr_measures_regr.smse, 69–85, 85, 87, 88
mlr_measures_regr.srho, 69–86, 86, 88
mlr_measures_regr.sse, 69–87, 87
mlr_measures_selected_features, 88
mlr_measures_time_both
(mlr_measures_elapsed_time), 67
mlr_measures_time_predict
(mlr_measures_elapsed_time), 67
mlr_measures_time_train
(mlr_measures_elapsed_time), 67
mlr_reflections, 109
mlr_reflections$default_measures, 9, 19
mlr_reflections$learner_predict_types, 20, 21, 27
mlr_reflections$learner_properties, 20,
mlr_reflections$measure_properties, 27
mlr_reflections$task_feature_types, 20, 21
mlr_reflections$task_properties, 122
mlr_reflections$task_types, 21
mlr_reflections$task_types$type, 121, 127
mlr_resamplings, 30, 35, 89, 91–97, 99, 105, 117, 119
mlr_resamplings_bootstrap, 90
mlr_resamplings_custom, 91
mlr_resamplings_cv, 92
mlr_resamplings_holdout, 93
mlr_resamplings_repeated_cv, 94
mlr_resamplings_subsampling, 96
mlr_sugar, 97
mlr_task_generators, 30, 35, 90, 97, 99, 105, 106–108, 127, 128
mlr_task_generators_2dnormal, 106
mlr_task_generators_friedman1, 106
mlr_task_generators_smiley, 107
mlr_task_generators_xor, 108
mlr_tasks, 30, 35, 90, 97, 98, 100–105, 120, 125, 126, 128
mlr_tasks_boston_housing, 99
mlr_tasks_german_credit, 100
mlr_tasks_iris, 101
mlr_tasks_mtcars, 101
mlr_tasks_pima, 102
mlr_tasks_sonar, 102
mlr_tasks_spam, 103
mlr_tasks_wine, 104
mlr_tasks_zoo, 104
msr (mlr_sugar), 97
msr(), 35
msrs (mlr_sugar), 97
paradox::ParamSet, 20, 21, 23, 98, 118, 127
ParamSet, 23
precision(), 43
predict.Learner, 108
PredictionClassif, 24, 109, 110, 111, 113
PredictionRegr, 25, 109, 110, 112, 113
R6::R6Class, 12, 15, 17, 18, 20, 24–26, 28–33, 89–93, 96, 98, 100–109, 111, 113, 116, 118, 120, 125, 127, 128
recall(), 43
regr.featureless, 30
regr.mse, 29
regr.rpart, 20, 26, 30
requireNamespace(), 21, 27, 127
resample, 114
resample(), 20, 22, 27, 28, 115–117
ResampleResult, 6, 11–13, 20, 22, 27, 28, 114, 115, 117
Resampling, 7, 9, 10, 12–15, 27, 89–97, 114, 116, 117, 119, 122
ResamplingBootstrap, 117
ResamplingRepeatedCV (mlr_resamplings_bootstrap), 90
ResamplingCustom (mlr_resamplings_custom), 91
ResamplingCV, 117, 118
ResamplingCV (mlr_resamplings_cv), 92
ResamplingHoldout, 118
ResamplingHoldout (mlr_resamplings_holdout), 93
ResamplingRepeatedCV (mlr_resamplings_repeated_cv), 94
Resamplings, 91–96, 119
ResamplingSubsampling (mlr_resamplings_subsampling), 96
rownames(), 18
rpart::rpart(), 33, 34
rse(), 83
rsmp (mlr_sugar), 97
rsmp(), 89, 90
rsmps (mlr_sugar), 97
setdiff(), 121
spam, 99
stats::cor(), 69, 86
stats::formula(), 123
stats::predict(), 108
Task, 7, 9, 10, 12–15, 22, 27, 28, 40, 88, 97–99, 110, 114, 116–119, 120, 122, 124–128
TaskClassif, 99–104, 111, 120, 125, 128
TaskGenerator, 97, 105–108, 127
TaskGenerator2DNormals
  (mlr_task_generators_2dnormals), 106
TaskGeneratorFriedman1
  (mlr_task_generators_friedman1), 106
TaskGenerators, 106–108
TaskGeneratorSmiley
  (mlr_task_generators_smiley), 107
TaskGeneratorXor
  (mlr_task_generators_xor), 108
TaskRegr, 99–101, 113, 120, 125, 126, 128
Tasks, 100–105
TaskSupervised, 99, 125, 126, 128
tgen (mlr_sugar), 97
tgen(), 105
tgens (mlr_sugar), 97
time_train, 26, 35
tsk (mlr_sugar), 97
tsk(), 98, 99
tsks (mlr_sugar), 97
union(), 121
xor, 105, 127