### mlr3 Dictionaries

Key-value store for sets of mlr objects. These are provided by mlr3:
- `mlr_tasks` - ML example tasks.
- `mlr_task_generators` - Example generators.
- `mlr_learners` - ML algorithms.
- `mlr_measures` - Performance measures.
- `mlr_resamplings` - Resampling strategies.

These dictionaries can be extended by loading extension packages. For example, by loading the `mlrlearners` package, the `mlr_learners` dictionary is extended with more learners.

#### Dictionary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>keys(pattern = NULL)</code></td>
<td>Returns all keys which match pattern. If NULL, all keys are returned.</td>
</tr>
<tr>
<td><code>get(key, ...)</code></td>
<td>Retrieves object by key and passes arguments &quot;...&quot; to the construction of the objects.</td>
</tr>
<tr>
<td><code>sget(keys, ...)</code></td>
<td>Retrieves objects by keys and passes named arguments &quot;...&quot; to the construction of the objects.</td>
</tr>
</tbody>
</table>

#### Data.table

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>as.data.table(Dictionary)</code></td>
<td>Lists objects with metadata.</td>
</tr>
</tbody>
</table>
Class: Resampling

Define partitioning of task into train and test sets. Creation:
resampling = rsmp(key, ...)”.

- holdout (ratio): Holdout-validation.
- cv(folds): k-fold cross-validation.
- repeated_cv (folds, repeats): Repeated k-fold cross-validation.
- subsampling (repeats, ratio): Repeated holdouts.
- Custom splits

resampling = rsmp("custom")
resampling$instantiate(task, train = list(c(1:10, 51:68, 101:111)), test = list(c(11:20, 61:78, 111:120)))

Returns a description of parameter settings.

resampling$param_set

Sets folds to 10.

resampling$param_set$values = list(folds = 10)

Returns a description of parameter settings.

Example

library(mlr3learners)

# task definition

task = tsk("pima")

# learner definition

learner = lrn("classif.rpart", predict_type = "prob")

# resampling definition

resampling = rsmp("cv", folds = 3)

# resampling instantiation

rr = resampling$instantiate(task, learner, resampling)

Resample

- Train-Predict: Score a learner on each train/test set.

rr = resample(task, learner, resampling)

Returns a ResampleResult container object.

rr$score(measures)

Returns a data.table of scores on test sets.

rr$aggregate(measures)

Gets aggregated performance scores as vector.

rr$filter(ites)

Filters to specific iterations.

Benchmarks

Compare learner(s) on task(s) with resampling(s).

design = benchmark_grid(tasks, learners, resamplings)

data.data.table of ResampleResult with scores.

bmr$score(measures)

Data.data.table of resampling iterations with scores.

bmr$filter(task_ids, learner_ids, resampling_ids)

Filter by task, learner and resampling.

Parallelization

The future framework is used for parallelization.

future: plan(backend)

Selects the parallelization backend for the current session. Parallelization is automatically applied to all levels (resampling, tuning andFeatSel).

Logging

1-grep is used for logging and progress output.

getoption("gpr.log_levels")

# got error warning info debug trace
# 100 200 300 400 500 600

Gets threshold levels. The default is 400.

lgr:get_logger("mlr3")$set_threshold("<level>")

Changes the log-level on a per-package basis.

mlr3viz

Provides visualization for mlr3 objects.

Examples:

BenchmarkResult (boxplot of performance measures, roc, prc)

Filter (barplot of filter scores)

PredictionClassif (Stacked barplot of true and estimated class labels, roc, prc)

PredictionRegr (xy scatterplot, histogram of residuals)

ResampleBoxplot (boxplot or histogram of performance measures, roc, prc)

TaskClassif (barplot of target, duo target-feature plot matrix, pairs feature plot matrix with color set to target)

TaskRegr (target, pairs)

TaskSurv (target, duo, pairs)

Error Handling and Encapsulation

Packages evaluate and callr can be used to encapsulate execution of$train() and$predict() to prevent stops in case of errors - useful for larger experiments. callr isolates the execution in a separate R sessions, guarding against segfaults.

learner$encapsulate = c(
  train = "evaluate",
predict = "callr")

 learner$errors

Returns the log of recorded errors.

learner$fallback = lrn(.key)

If learner fails, a fallback learner is used to generate predictions. Use a robust fallback, e.g. a "featureless" learner.

Resources

- mlr-org on GitHub (https://github.com/mlr-org)
- mlr3learners R package (https://github.com/mlr-org/mlr3learners)
- mlr3learners organization (https://github.com/mlr3learners)
- mlr3gallery use cases (https://mlr3gallery.mlr-org.com)

mlr-org.com, cheatsheets.mlr-org.com