Machine learning with mlr3::CHEAT SHEET

Class Overview

The mlr3 package builds on R6 classes and provides the essential building blocks of a machine learning workflow.

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 **mlr3learners** package, the `mlr_learners` dictionary is extended with more learners. Syntactic sugar functions retrieve objects from dictionaries, set hyperparameters and assign fields in one go e.g. `lrn("classif.rpart", cp = 0.1)`.

```
Dictionary$keys(pattern = NULL)
```

Returns all keys which match pattern. If NULL, all keys are returned.

```
Dictionary$get(key, ...)
```

Retrieves object by key and passes arguments "…" to the construction of the objects.

```
Dictionary$mget(keys, ...)
```

Retrieves objects by keys and passes named arguments "…" to the construction of the objects.

```
as.data.table(Dictionary)
```

Lists objects with metadata.

Class: Task

Stores data and metadata. `x` can be a `data.table`, `target` points to y-column by name.

```
task = as_task_regr(backend, target)
```

Create task for regression or classification.

```
task = tsk(.key)
```

Sugar to get example task from `mlr_tasks`:

- Twoclass: `german_credit`, `pima`, `sonar`, `spam`
- Multiclass: `iris`, `wine`, `zoo`
- Regression: `boston_housing`, `mtcars`

Print the `mlr_tasks` dictionary for more.

```
task$positive = "<positive_class>"
```

Set positive class for binary classification.

Column Roles

Column roles affect the behavior of the task for different operations. Set with
taskcol_roles<role> = "<column_name>":

- `feature` - Regular features.
- `target` - Target variable.
- `name` - Labels for plots.
- `group` - Groups for block resampling.
- `stratum` - Stratification variables.
- `weight` - Observation weights.

Data Operations

```
task$select(cols)
```

Subsets the task based on feature names.

```
task$filter(rows)
```

Subsets the task based on row ids.

```
task$cbind(data) / task$rbind(data)
```

Adds additional columns / rows.

```
task$rename(from, to)
```

Rename columns.

Class: Learner

Wraps learners from R with a unified interface.

```
learner = lrn(.key, ...)
```

Get learner by `.key` (from `mlr_learners`) and construct the learner with specific hyperparameters and settings "…" in one go. github.com/mlr-org/mlr3learners (R package) and github.com/mlr3learners (GitHub organization) hold all available learners.

```
learner$param_set
```

Returns description of hyperparameters.

```
learner$param_set$values = list(id = value)
```

Change the current hyperparameter values by assigning a named `list(id = value)` to the `$values` field. This overwrites all previously set parameters.

```
learner$param_set$values$<id> = <value>
```

Update a single hyperparameter.

```
learner$predict_type = "<type>"
```

Changes/sets the output type of the prediction. For classification, `"response"` means class labels, `"prob"` means posterior probabilities. For regression, `"response"` means numeric response, `"se"` extracts the standard error.

Example

```
task = tsk("sonar")
learner = lrn("classif.rpart")

train_set = sample(task$nrow, 0.8 * task$nrow)
test_set = setdiff(seq_len(task$nrow), train_set)

learner$train(task, row_ids = train_set)

prediction = learner$predict(task, row_ids = test_set)
prediction$score()
## > classif.ce
## > 0.2619048
```

Train & Predict

```
learner$train(task, row_ids)
```

Train on (selected) observations.

```
learner$model
```

The resulting model is stored in the `$model` slot of the `learner`.

```
prediction = learner$predict(task, row_ids)
```

Predict on (selected) observations.

Measures & Scoring

```
measure = msr(.key)
```

Get measure by `.key` from `mlr_measures:

- `classif.ce` - Classification error.
- `classif.auc` - AUROC.
- `regr.rmse` - Root mean square error.

Print `mlr_measures` for all measures.

```
prediction$score(measures)
```

Calculate performance with one or more measures.

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.
- `bootstrap` (`repeats`, `ratio`) Out-of-bag bootstrap.
- Custom splits

```
resampling = rsmp("custom")
resampling$instantiate(task,
  train = list(c(1:10, 51:60, 101:110)),
  test = list(c(11:20, 61:70, 111:120)))
```

```
resampling$param_set
```

Returns a description of parameter settings.

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

Sets folds to 10.

```
task$col_roles$stratum = "<column_names>"
```

Sets stratification variables.

```
task$col_roles$group = "<column_name>"
```

Sets group variable.

```
resampling$instantiate(task)
```

Perform splitting and define index sets.

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(iters)
```

Filters to specific iterations.

Example

```
library(mlr3learners)
task = tsk("pima")
learner = lrn("classif.rpart", predict_type = "prob")
measure = msr("classif.ce")
resampling = rsmp("cv", folds = 3L)
resampling$instantiate(task)
rr = resample(task, learner, resampling)
as.data.table(rr)[, list(resampling, iteration, prediction)]
## >          resampling  iteration            prediction
## > 1: <ResamplingCV[19]>         1 <PredictionClassif[19]>
## > 2: <ResamplingCV[19]>         2 <PredictionClassif[19]>
## > 3: <ResamplingCV[19]>         3 <PredictionClassif[19]>
rr$aggregate(measure)
## > classif.ce
## >  0.2239583
learners = lrns(c("classif.rpart", "classif.ranger"))
tasks = tsks(c("sonar", "spam"))
resampling = rsmp("cv", folds = 3L)
design = benchmark_grid(tasks, learners,resampling)
bmr = benchmark(design)
bmr
## >                       learner         resampling   iteration
## >  1: <LearnerClassifRpart[33]>  <ResamplingCV[19]>          1
## >  2: <LearnerClassifRpart[33]>  <ResamplingCV[19]>          2
## >  3: <LearnerClassifRpart[33]>  <ResamplingCV[19]>          3
## >  4: <LearnerClassifRanger[33]> <ResamplingCV[19]>          1
bmr$aggregate()[, list(nr, resample_result, task_id, learner_id, classif.ce)]
## >   nr      resample_result task_id     learner_id classif.ce
## > 1:  1 <ResampleResult[21]>   sonar  classif.rpart 0.30276052
## > 2:  2 <ResampleResult[21]>   sonar classif.ranger 0.17308489
## > 3:  3 <ResampleResult[21]>    spam  classif.rpart 0.09997865
## > 4:  4 <ResampleResult[21]>    spam classif.ranger 0.04868526
```

Results are stored as a `data.table`. `BenchmarkResult` contains a `ResampleResult` object for each task-learner-resampling combination which in turn contain a `Prediction` object for each resampling iteration.

Benchmark

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

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

Creates a cross-join datatable with list-columns. Can also be set up manually for full control.

```
bmr = benchmark(design)
```

Returns a `BenckmarkResult` container.

```
bmr$aggregate(measures)
```

`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.

```
bmr$combine(bmr)
c(bmr, bmr1) # alternative S3 method
```

Merge other `BenchmarkResult`.

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 and FeatSel).

Logging

`lgr` is used for logging and progress output.

```
getOption("lgr.log_levels")
## > fatal error  warn  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. Creation: `mlr3viz::autoplot(object, type)`

- `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)
- `ResampleResult` (boxplot or `histogram` of performance measures, `roc`, `prc`)
- `TaskClassif` (barplot of `target`, duo target-features 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

- mlr3book (https://mlr3book.mlr-org.com)
- 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/)