Machine learning with mlr3::CHEAT SHEET

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

mlr Dictionaries

Key-value store for sets of mlr objects. These are provided by mlr:
- 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 eg 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$get(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.

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

Create task for regression or classification.

```r
task = task(.key)
```

Sugar to get example task from mlr_tasks:

- Two-class: german_credit, pima, sonar, spam
- Multiclass: iris, wine, zoo
- Regression: boston_housing, mtcars

Print the mlr_tasks dictionary for more.

```r
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 `task$col_roles<$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

```r
task$select(cols)
```
Subsets the task based on feature names.

```r
task$filter(rows)
```
Subsets the task based on row ids.

```r
task$bind(data) / task$rbind(data)
```
Adds additional columns / rows.

```r
task$rename(from, to)
```
Rename columns.

Class: Learner
Wraps learners from R with a unified interface.

```r
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.

```r
learner$param_set
```
Returns description of hyperparameters.

```r
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.

```r
learner$predict_type = type
```
Update a single hyperparameter.

```r
learner$set(type = "<type>"
```
Changes/set 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.

Train & Predict

```r
learner$train(task, row_ids)
```
Train on (selected) observations.

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

```r
prediction = learner$predict(task, row_ids)
```
Predict on (selected) observations.

Measures & Scoring

```r
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.

```r
prediction$score(measures)
```
Calculate performance with one or more measures.

Example

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

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

learner$train(task, row_ids = train_set)
prediction = learner$predict(task, row_ids = test_set)
prediction$score()
```
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(\(\text{custom}\))
resampling$initialize(task, train = list(c(1:10, 51:60, 181:181:181)), test = list(c(11:28, 61:78, 111:128)))

resampling$param_set

Returns a description of parameter settings.

Example

library(mlr3learners)
task = task$create_task(task_id = "pima")
learner = learner$create_learner(learner_id = "classif.rpart")
resampling = resample(task, learner, resampling)
resampling$instantiate(task)

Perform splitting and define index sets.

Resample

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

\[
rr = \text{resample}(\text{task}, \text{learner}, \text{resampling})
\]

Returns a ResampleResult container object.

\[
rr$score(\text{measures})
\]

Returns a data.table of scores on test sets.

\[
rr$aggregate(\text{measures})
\]

Gets aggregated performance scores as vector.

\[
rr$filter(\text{iters})
\]

Filters to specific iterations.

Benchmark

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

\[
\text{design} = \text{benchmark_grid}(\text{tasks}, \text{learners}, \text{resamplings})
\]

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

\[
\text{bar} = \text{benchmark}(\text{design})
\]

Returns a BenchamrkResult container.

\[
\text{bar}@aggregate(\text{measures})
\]

Data.data.table of ResampleResult with scores.

\[
\text{bar}$score(\text{measures})
\]

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.

\[
\text{learner}$encapsulate = \text{callr}()
\]

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

Parallelization

The future framework is used for parallelization.

\[
\text{future} = \text{plan}(\text{backend})
\]

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

Logging

1gr is used for logging and progress output.

\[
\text{getOption}(\text{"1gr.log_levels"})
\]

Gets threshold levels. The default is 400.

\[
\text{1gr}$get_logger(\text{"mldr"})@set_threshold(\"level\")
\]

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

mlr-org.com, cheatsheets.mlorg.com