### Machine learning with mlr3::CHEAT SHEET

#### Class Overview

The package provides a set of R6 classes which allow to (a) define general feature selection instances and (b) run algorithms which optimize on these, (a) is called a FSelectInstanceSingleCrit or FSelectInstanceMultiCrit, which define a blackbox optimization function that maps feature subsets to resampled performance values for arbitrary performance measures.

#### mlr3 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 mlr_learners 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.

- **get(key, ...)**

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

- **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. backend can be a data.table, target points to column by name.

```r
task = TaskRegr$new(backend, target)
```

Create task for regression or classification.

```r
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.

```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

- **select(cols)**

  Subsets the task based on feature names.

- **filter(rows)**

  Subsets the task based on row ids.

- **bind(data) / task$bind(data)**

  Adds additional columns / rows.

- **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.

- **param_set**

  Returns description of hyperparameters.

```r
learner$param_set
```

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

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

Update a single hyperparameter.

```r
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

```r
task = tsk("sonar")
learner = lrn("classif.rpart")
train_set = sample(task$row, 0.8 * task$row)
test_set = setdiff(task$row, train_set)
learner$predict(task, row_ids = train_set)
set.seed(1)
prediction = learner$predict(task, row_ids = test_set)
```

Calculate performance with one or more measures.

#### Train & Predict

- **train(task, row_ids)**

  Train on (selected) observations.

- **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.

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

  Calculate performance with one or more measures.
Perform splitting and define index sets.

```
resampling = rsmp("custom")
resampling$set_train(test = list(c(1:18, 51:68, 81:118)), train = list(c(11:28, 61:78, 111:128)))
```

Returns a description of parameter settings.

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

Sets folds to 10.

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

Sets stratification variables.

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

Sets group variable.

```
resampling$initialize(task)
```

Performs splitting and defines index sets.

---

**Example**

```
library(mlr3learners)
task = tsk("pima")
learner = lrn("classif.ranger", product_type = "prp")
measure = msr("classif.ce")
resampling = rsmp("cv", folds = 3L)
resampling$initialize(task)
rr = resample(task, learner, resampling)
rr$score(measures)
```

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(iter)
```

Filters to specific iterations.

---

**Benchmarks**

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.

```
bar = benchmark(design)
```

Returns a BenchmarkResult container.

```
bar$aggregate(measures)
```

Data table of ResampleResult with scores.

```
bar$score(measures)
```

Data table of resampling iterations with scores.

```
bar$filter(task_ids, learner_ids, resampling_ids)
```

Filter by task, learner and resampling.

```
bar$combine(bar)
```

Merge other BenchmarkResult.

```
future::plan(backend)
```

Selects the parallelization backend for the current session.

Parallelization

The future framework is used for parallelization.

Logging

1gr is used for logging and progress output.

```
get_logger("1gr.log_levels")
```

Gets threshold levels. The default is 400.

```
1gr$set_logger("mlr3")$set_threshold("<level>")
```

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

---

**mlr3viz**

Provides visualization for mlr3 objects.

```
resampling = rsmp(.key, ...)
```

Benchmarks

- 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**

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