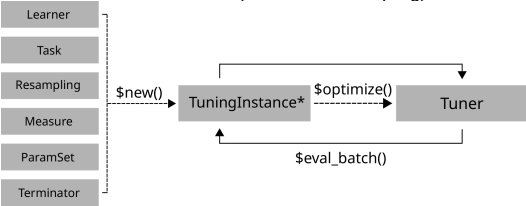


Hyperparameter Tuning with mlr3tuning::CHEAT SHEET

Class Overview

The package provides a set of R6 classes which allow to (a) define general hyperparameter (HP) tuning instances, i.e., the black-box objective that maps HP configurations (HPCs) to resampled performance values; (b) run black-box optimizers; (c) combine learners with tuners (for nested resampling).



ParamSet - Parameters and Ranges

Scalar doubles, integers, factors or logicals are combined to define a multivariate search space (SS).

```
tune_ps = ps(
  <id> = p_int(lower, upper),
  <id> = p_dbl(lower, upper),
  <id> = p_dct(levels),
  <id> = p_lgl())
```

id is identifier. lower/upper ranges, levels categories.

```
learner = lrn("classif.rpart",
  cp = to_tune(0.001, 0.1, logscale = TRUE),
  minsplit = to_tune(1, 10))
learner$param_set$search_space() # only for inspection
```

Or, use to_tune() to set SS for each param in Learner. SS is auto-generated when learner is tuned. logscale = TRUE creates SS on log-scale.

Terminators - When to stop

Construction: trm(.key, ...)

- evals (n_evals)
After iterations.
- run_time (secs)
After training time.
- clock_time (stop_time)
At given timepoint.
- perf_reached (level)
After performance was reached.
- stagnation (iters, threshold)
After performance stagnated.
- combo (list_of_terms, any=TRUE)
Combine terminators with AND or OR.

```
as.data.table(mlr_terminators) # list all
```

TuningInstance* - Search Scenario

Evaluator and container for resampled performances of HPCs during tuning. The main (internal) function eval_batch(xdt) calls benchmark() to evaluate a table of HP configurations. Also stores archive of all evaluated experiments and the final result.

```
instance = TuningInstanceSingleCrit$new(task,
  learner, resampling, measure, terminator, tune_ps)
```

store_benchmark_result = TRUE to store resampled evals and store_models = TRUE for fitted models.

Example

```
# optimize HPs of RBF SVM on logscale
learner = lrn("classif.svm", kernel = "radial", type = "C-classification")
ss = ps(
  cost = p_dbl(1e-4, 1e4, logscale = TRUE),
  gamma = p_dbl(1e-4, 1e4, logscale = TRUE))
evals = trm("evals", n_evals = 20)
instance = TuningInstanceSingleCrit$new(task, learner, resampling, measure, evals, ss)
tuner = tnr("random_search")
tuner$optimize(instance)
instance$result
```

Use TuningInstanceMultiCrit for multi-criteria tuning.

Tuner - Search Strategy

Generates HPCs and passes to tuning instance for evaluation until termination. Creation: tnr(.key, ...)

- grid_search (resolution, batch_size)
Grid search.
- random_search (batch_size)
Random search.
- design_points (design)
Search at predefined points.
- random_search (batch_size)
Random search.
- nloptr (algorithm)
Non-linear optimization.
- gensa (smooth, temperature)
Generalized Simulated Annealing.
- irace
Iterated racing.

```
as.data.table(mlr_tuners) # list all
```

Logging and Parallelization

```
lgr::get_logger("bbotk")$set_threshold("<level>")
```

Change log-level only for mlr3tuning.

```
future::plan(strategy)
```

Sets the parallelization backend. Speeds up tuning by running iterations in parallel.

Execute Tuning

```
tuner$optimize(instance)
```

```
as.data.table(instance$archive)
```

Returns all evaluated configurations and their resampling results. The x_domain_* columns contain HP values after the transformation.

Example

```
as.data.table(instance$archive)
## > cost gamma classif.ce uhash x_domain_cost x_domain_gamma
## > 1: 3.13 5.55 0.56 b8744... 3.13 5.55
## > 2: -1.94 1.32 0.10 f5623... -1.94 1.32
```

```
instance$result
```

Returns list with optimal configurations and estimated performance.

```
learner$param_set$values =
  instance$result_learner_param_vals
```

Set optimized HP in Learner.

Example

```
learner = lrn("classif.svm", type = "C-classification", kernel = "radial",
  cost = to_tune(1e-4, 1e4, logscale = TRUE))
instance = tune(method = "grid_search", task = tsk("iris"), learner = learner,
  resampling = rsm("holdout"), measure = msr("classif.ce"), resolution = 5)
```

Use tune()-shortcut.

AutoTuner - Tune before Train

Wraps learner and performs integrated tuning.

```
at = AutoTuner$new(learner, resampling, measure,
  terminator, tuner)
```

Inherits from class Learner. Training starts tuning on the training set. After completion the learner is trained with the "optimal" configuration on the given task.

```
at$train(task)
at$predict(task, row_ids)
```

```
at$learner
```

Returns tuned learner trained on full data set.

```
at$tuning_result
```

Access tuning result.

```
at = auto_tuner(method = "grid_search", learner,
  resampling, measure, term_evals = 20)
```