Package 'mlr3hyperband'

October 26, 2020

Title Hyperband for 'mlr3'

Version 0.1.0

Description Implements hyperband method for hyperparameter tuning. Various termination criteria can be set and combined. The class 'AutoTuner' provides a convenient way to perform nested resampling in combination with 'mlr3'. The hyperband algorithm was proposed by Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh and Ameet Talwalkar (2018) <arXiv:1603.06560>.

License LGPL-3

URL https://mlr3hyperband.mlr-org.com, https://github.com/mlr-org/mlr3hyperband

BugReports https://github.com/mlr-org/mlr3hyperband/issues

Depends R (>= 3.1.0)

Imports bbotk (>= 0.2.0), checkmate (>= 1.9.4), data.table, lgr, mlr3 (>= 0.7.0), mlr3misc, mlr3tuning (>= 0.2.0), paradox, R6

Suggests emoa, mlr3learners, mlr3pipelines, rpart, testthat, xgboost

Encoding UTF-8

NeedsCompilation no

RoxygenNote 7.1.1

Collate 'TunerHyperband.R' 'nds_selection.R' 'bibentries.R' 'zzz.R'

Author Marc Becker [aut, cre] (<https://orcid.org/0000-0002-8115-0400>), Sebastian Gruber [aut] (<https://orcid.org/0000-0002-8544-3470>), Jakob Richter [aut] (<https://orcid.org/0000-0003-4481-5554>), Julia Moosbauer [aut] (<https://orcid.org/0000-0002-0000-9297>), Bernd Bischl [aut] (<https://orcid.org/0000-0001-6002-6980>)

Maintainer Marc Becker <marcbecker@posteo.de>

Repository CRAN

Date/Publication 2020-10-26 10:40:03 UTC

R topics documented:

`mlr3hyperband-package` *mlr3hyperband: Hyperband for 'mlr3'*

Description

Implements hyperband method for hyperparameter tuning. Various termination criteria can be set and combined. The class 'AutoTuner' provides a convenient way to perform nested resampling in combination with 'mlr3'. The hyperband algorithm was proposed by Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh and Ameet Talwalkar (2018) <arXiv:1603.06560>.

Author(s)

Maintainer: Marc Becker <marcbecker@posteo.de> (ORCID)

Authors:

- Sebastian Gruber <gruber_sebastian@t-online.de> (ORCID)
- Jakob Richter <jakob1richter@gmail.com> (ORCID)
- Julia Moosbauer <ju.moosbauer@googlemail.com> (ORCID)
- Bernd Bischl <bernd_bischl@gmx.net> (ORCID)

See Also

Useful links:

- https://mlr3hyperband.mlr-org.com
- https://github.com/mlr-org/mlr3hyperband
- Report bugs at https://github.com/mlr-org/mlr3hyperband/issues

`mlr_tuners_hyperband` *Tuner using the Hyperband algorithm*

Description

`TunerHyperband` class that implements hyperband tuning. Hyperband is a budget oriented-procedure, weeding out suboptimal performing configurations early in a sequential training process, increasing tuning efficiency as a consequence.

For this, several brackets are constructed with an associated set of configurations for each bracket. Each bracket as several stages. Different brackets are initialized with different amounts of configurations and different budget sizes. To get an idea of how the bracket layout looks like for a given argument set, please have a look in the `details`.

To identify the budget for evaluating hyperband, the user has to specify explicitly which hyperparameter of the learner influences the budget by tagging a single hyperparameter in the paradox::ParamSet with `"budget"`. An alternative approach using subsampling and pipelines is described below.

Naturally, hyperband terminates once all of its brackets are evaluated, so a bbotk::Terminator in the tuning instance acts as an upper bound and should be only set to a low value if one is unsure of how long hyperband will take to finish under the given settings.

Details

This sections explains the calculation of the constants for each bracket. A small overview will be given here, but for more details please check out the original paper (see `references`). To keep things uniform with the notation in the paper (and to safe space in the formulas), `R` is used for the upper budget that last remaining configuration should reach. The formula to calculate the amount of brackets is `floor(log(R,eta)) + 1`. To calculate the starting budget in each bracket, use `R * eta^(-s)`, where `s` is the maximum bracket minus the current bracket index. For the starting configurations in each bracket it is `ceiling((B/R) * ((eta^s)/(s+1)))`, with B = (bracket amount) * R. To receive a table with the full brackets layout, load the following function and execute it for the desired `R` and `eta`.

```
hyperband_brackets = function(R, eta) {

  result = data.frame()
  smax = floor(log(R, eta))
  B = (smax + 1) * R

  # outer loop - iterate over brackets
  for (s in smax:0) {

    n = ceiling((B/R) * ((eta^s)/(s+1)))
    r = R * eta^(-s)

    # inner loop - iterate over bracket stages
    for (i in 0:s) {
```

```
      ni = floor(n * eta^(-i))
      ri = r * eta^i
      result = rbind(result, c(smax - s + 1, i + 1, ri, ni))
    }
  }

  names(result) = c("bracket", "bracket_stage", "budget", "n_configs")
  return(result)
}

hyperband_brackets(R = 81L, eta = 3L)
```

Parameters

`eta` `numeric(1)`
: Fraction parameter of the successive halving algorithm: With every step the configuration budget is increased by a factor of `eta` and only the best `1/eta` configurations are used for the next stage. Non-integer values are supported, but `eta` is not allowed to be less or equal 1.

`sampler` paradox::Sampler
: Object defining how the samples of the parameter space should be drawn during the initialization of each bracket. The default is uniform sampling.

Archive

The mlr3tuning::ArchiveTuning holds the following additional columns that are specific to the hyperband tuner:

- `bracket` (`integer(1)`)
  The console logs about the bracket index are actually not matching with the original hyperband algorithm, which counts down the brackets and stops after evaluating bracket 0. The true bracket indices are given in this column.
- `bracket_stage` (integer(1))
  The bracket stage of each bracket. Hyperband starts counting at 0.
- `budget_scaled` (`numeric(1)`)
  The intermediate budget in each bracket stage calculated by hyperband. Because hyperband is originally only considered for budgets starting at 1, some rescaling is done to allow budgets starting at different values. For this, budgets are internally divided by the lower budget bound to get a lower budget of 1. Before the learner receives its budgets for evaluation, the budget is transformed back to match the original scale again.
- `budget_real` (`numeric(1)`)
  The real budget values the learner uses for evaluation after hyperband calculated its scaled budget.
- `n_configs` (`integer(1)`)
  The amount of evaluated configurations in each stage. These correspond to the `r_i` in the original paper.

Hyperband without learner budget

Thanks to **mlr3pipelines**, it is possible to use hyperband in combination with learners lacking a natural budget parameter. For example, any mlr3::Learner can be augmented with a mlr3pipelines::PipeOp operator such as mlr3pipelines::PipeOpSubsample. With the subsampling rate as budget parameter, the resulting mlr3pipelines::GraphLearner is fitted on small proportions of the mlr3::Task in the first brackets, and on the complete Task in last brackets. See examples for some code.

Custom sampler

Hyperband supports custom paradox::Sampler object for initial configurations in each bracket. A custom sampler may look like this (the full example is given in the `examples` section):

```
# - beta distribution with alpha = 2 and beta = 5
# - categorical distribution with custom probabilities
sampler = SamplerJointIndep$new(list(
  Sampler1DRfun$new(params[[2]], function(n) rbeta(n, 2, 5)),
  Sampler1DCateg$new(params[[3]], prob = c(0.2, 0.3, 0.5))
))
```

Runtime scaling w.r.t. the chosen budget

The calculation of each bracket currently assumes a linear runtime in the chosen budget parameter is always given. Hyperband is designed so each bracket requires approximately the same runtime as the sum of the budget over all configurations in each bracket is roughly the same. This will not hold true once the scaling in the budget parameter is not linear anymore, even though the sum of the budgets in each bracket remains the same. A basic example can be viewed by calling the function `hyperband_brackets` below with the arguments `R = 2` and `eta = 2`. If we run a learner with O(budget^2) time complexity, the runtime of the last bracket will be 33% longer than the first bracket (time of bracket 1 = 2 * 1^2 + 2^2 = 6; time of bracket 2 = 2 * 2^2 = 8). Of course, this won't break anything, but it should be kept in mind when applying hyperband. A possible adaption would be to introduce a trafo, like it is shown in the `examples`.

Logging

When loading the mlr3hyperband package, two loggers based on the lgr package are made available. One is called `mlr3`, the other `bbotk`. All `mlr3` methods log into the `mlr3` logger. All optimization methods form the packags bbotk, mlr3tuning and mlr3hyperband log into the `bbotk` logger. To hide the mlr3 logging messages run:

```
lgr::get_logger("mlr3")$set_threshold("warn")
```

Super class

`mlr3tuning::Tuner` -> `TunerHyperband`

Methods

Public methods:

- `TunerHyperband$new()`

- TunerHyperband$clone()

Method `new()`**:** Creates a new instance of this R6 class.

Usage:

```
TunerHyperband$new()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TunerHyperband$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Source

Li L, Jamieson K, DeSalvo G, Rostamizadeh A, Talwalkar A (2018). "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization." *Journal of Machine Learning Research*, **18**(185), 1-52. https://jmlr.org/papers/v18/16-558.html.

Examples

```
library(mlr3)
library(mlr3learners)
library(paradox)
library(mlr3tuning)
library(mlr3hyperband)

# Define hyperparameter and budget parameter for tuning with hyperband
ps = ParamSet$new(list(
  ParamInt$new("nrounds", lower = 1, upper = 4, tag = "budget"),
  ParamDbl$new("eta", lower = 0, upper = 1),
  ParamFct$new("booster", levels = c("gbtree", "gblinear", "dart"))
))

# Define termination criterion
# Hyperband terminates itself
terminator = trm("none")

# Create tuning instance
inst = TuningInstanceSingleCrit$new(
  task = tsk("iris"),
  learner = lrn("classif.xgboost"),
  resampling = rsmp("holdout"),
  measure = msr("classif.ce"),
  search_space = ps,
  terminator = terminator,
)

# Load tuner
tuner = tnr("hyperband", eta = 2L)
```

```
# Trigger optimization
tuner$optimize(inst)

# Print all evaluations
inst$archive$data()
```

`nds_selection` *Best points w.r.t. non dominated sorting with hypervolume contrib.*

Description

Select best subset of points by non dominated sorting with hypervolume contribution for tie breaking. Works on an arbitrary dimension of size two or higher. Returns a vector of indices of selected points.

Value

`integer()`

Parameters

`points matrix()`
: Numeric matrix with each column corresponding to a point.

`n_select integer(1)`
: Amount of points to select.

`ref_point integer()`
: Reference point for hypervolume.

`minimize logical()`
: Should the ranking be based on minimization? (Single bool for all dimensions, or vector of bools with each entry corresponding to each dimension).

Index