Hyperbandr Tutorial

Niklas

February 21, 2018

Hyperbandr Package

This is an R6 implementation of the original hyperband algorithm https://arxiv.org/abs/1603.06560.

R6 is an encapsulated object oriented system akin to those in Java or C++, where objects contain methods in addition to data, and those methods can modify objects directly (unlike S3 and S4 which are both functional object-oriented systems, where class methods are separate from objects, and objects are not mutable).

Essentially, that means that we obtain a very generic implementation, which is working with every other R package (as long the algorithm meets the requirements of hyperband).

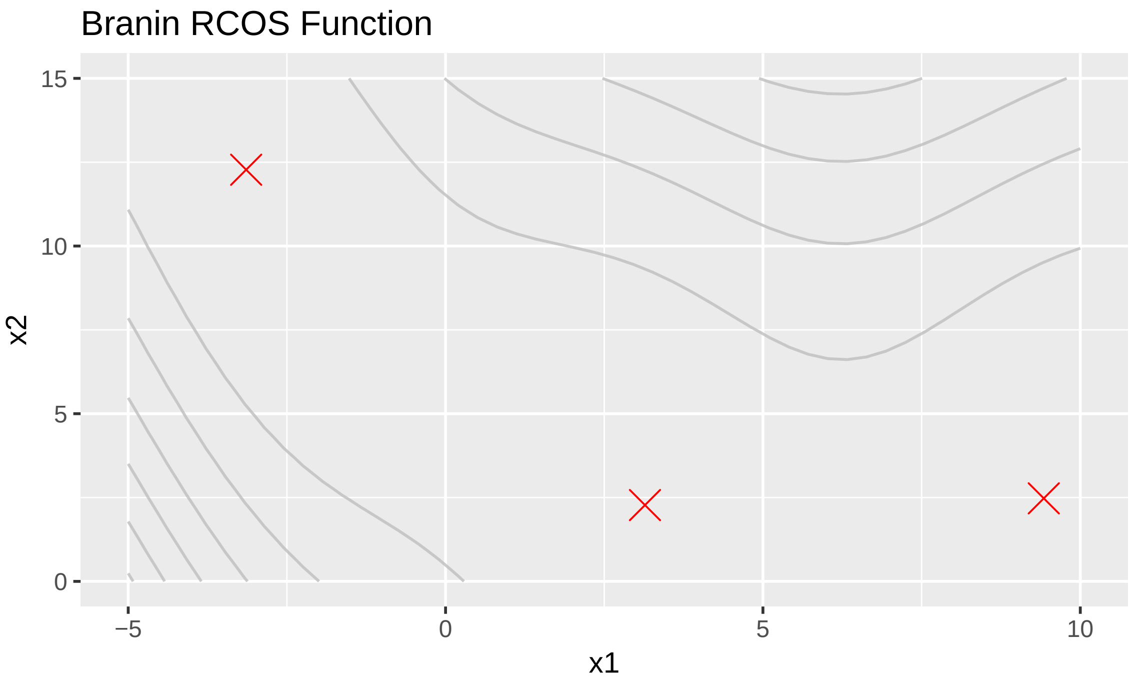
Simple example: optimize the branin function

```r
library("smoof")
library("data.table")
library("ggplot2")
```

```r
problem = makeBraninFunction()

# the branin function has 3 global minima (red crosses)
opt = data.table(x1 = getGlobalOptimum(problem)$param$x1,
                 x2 = getGlobalOptimum(problem)$param$x2)

(vis = autoplot(problem) + geom_point(data = opt, aes(x = x1, y = x2),
                                      shape = 4, colour = "red", size = 5))
```



We treat the value of x_1 as our "configuration" and try to find the optimal value for our "hyperparameter" x_2 (reminder: in hyperband, we have to sample configurations for each bracket)

So in order to apply **hyperbandr** on that problem, we need to define a hyperparameter space and four specific functions.

Configuration Space

As our very first step, we need to define the hyperparameter space to sample our configurations from. We want to obtain random values between -5 and approximately 10.1 (our x_1 axis..)

```r
# we use the makeParamSet function from the ParamHelpers package
configSpace = makeParamSet(
    makeNumericParam(id = "x1", lower = -5, upper = 10.1))
```

Function 1: the sampling function

Now we need a function to sample configurations from the configuration space

```r
# par.set: the parameter space to sample from
# n.configs: the amount of configs to sample
sample.fun = function(par.set, n.configs) {
  sampleValues(par = par.set, n = n.configs)
}
```

Function 2: the initialization function

This function takes a config and samples a corresponding value of x_2 in order to initialize the model

```r
# r: initial budget used for the initialization
# config: one configuration sampled by sample.fun
init.fun = function(r, config) {
  x1 = unname(unlist(config))
  x2 = runif(1, 0, 15)
  mod = c(x1, x2)
  return(mod)
}
```

Function 3: the training function

To train our model, we simply sample values from a normal distribution and add or subtract them from our current x_2. If the performance improves, we keep the model, else we discard it and keep the old one.

```r
# mod: the model to train
# budget: number of iterations to train the model for
train.fun = function(mod, budget) {
  for(i in seq_len(budget)) {
    mod.new = c(mod[[1]], mod[[2]] + rnorm(1, sd = 3))
    if(performance.fun(mod.new) < performance.fun(mod))
      mod = mod.new
  }
  return(mod)
}
```

Function 4: the performance function

Finally, we define a function to evaluate the performance of each model

```
# model: the model to evaluate
performance.fun = function(model) {
  problem(c(model[[1]], model[[2]]))
}
```

Apply hyperbandr (since the problem to optimize here is very easy, the execution will only take 1-2 seconds)

```
library("R6")
library("devtools")
load_all()
```

```
hyperhyper = hyperband(
  max.perf = FALSE,
  max.ressources = 81,
  prop.discard = 3,
  id = "branin",
  par.set = configSpace,
  sample.fun =  sample.fun,
  train.fun = train.fun,
  performance.fun = performance.fun)
```

```
## Beginning with bracket 4
## Iteration 0, with 81 Algorithms left (Budget: 1)
## Iteration 1, with 27 Algorithms left (Budget: 4)
## Iteration 2, with 9 Algorithms left (Budget: 13)
## Iteration 3, with 3 Algorithms left (Budget: 40)
## Iteration 4, with 1 Algorithms left (Budget: 121)
## Beginning with bracket 3
## Iteration 0, with 34 Algorithms left (Budget: 3)
## Iteration 1, with 11 Algorithms left (Budget: 12)
## Iteration 2, with 3 Algorithms left (Budget: 39)
## Iteration 3, with 1 Algorithms left (Budget: 120)
## Beginning with bracket 2
## Iteration 0, with 15 Algorithms left (Budget: 9)
## Iteration 1, with 5 Algorithms left (Budget: 36)
## Iteration 2, with 1 Algorithms left (Budget: 117)
## Beginning with bracket 1
## Iteration 0, with 8 Algorithms left (Budget: 27)
## Iteration 1, with 1 Algorithms left (Budget: 108)
## Beginning with bracket 0
## Iteration 0, with 1 Algorithms left (Budget: 81)
```

Let us inspect the results: we obtain a list of 5 R6 objects.

```
hyperhyper
```

```
## [[1]]
## <bracket>
##   Public:
##     B: 405
##     clone: function (deep = FALSE)
##     configurations: list
##     filterTopKModels: function (k)
##     getBudgetAllocation: function ()
##     getNumberOfModelsToSelect: function ()
##     getPerformances: function ()
##     getTopKModels: function (k)
##     id: branin
##     initialize: function (max.perf, max.ressources, prop.discard, s, B, id, par.set,
##     iteration: 4
##     max.perf: FALSE
##     max.ressources: NULL
##     models: list
##     n.configs: 1
##     par.set: NULL
##     printState: function ()
##     prop.discard: 3
##     r.config: 1
##     run: function ()
##     s: 4
##     sample.fun: NULL
##     step: function ()
##
## [[2]]
## <bracket>
##   Public:
##     B: 405
##     clone: function (deep = FALSE)
##     configurations: list
##     filterTopKModels: function (k)
##     getBudgetAllocation: function ()
##     getNumberOfModelsToSelect: function ()
##     getPerformances: function ()
##     getTopKModels: function (k)
##     id: branin
##     initialize: function (max.perf, max.ressources, prop.discard, s, B, id, par.set,
##     iteration: 3
##     max.perf: FALSE
##     max.ressources: NULL
##     models: list
##     n.configs: 1
##     par.set: NULL
##     printState: function ()
##     prop.discard: 3
##     r.config: 3
##     run: function ()
##     s: 3
##     sample.fun: NULL
##     step: function ()
##
## [[3]]
## <bracket>
##   Public:
##     B: 405
##     clone: function (deep = FALSE)
##     configurations: list
##     filterTopKModels: function (k)
##     getBudgetAllocation: function ()
##     getNumberOfModelsToSelect: function ()
##     getPerformances: function ()
##     getTopKModels: function (k)
##     id: branin
##     initialize: function (max.perf, max.ressources, prop.discard, s, B, id, par.set,
##     iteration: 2
##     max.perf: FALSE
##     max.ressources: NULL
##     models: list
##     n.configs: 1
##     par.set: NULL
##     printState: function ()
##     prop.discard: 3
##     r.config: 9
##     run: function ()
##     s: 2
##     sample.fun: NULL
##     step: function ()
##
## [[4]]
## <bracket>
##   Public:
##     B: 405
##     clone: function (deep = FALSE)
##     configurations: list
##     filterTopKModels: function (k)
##     getBudgetAllocation: function ()
##     getNumberOfModelsToSelect: function ()
##     getPerformances: function ()
##     getTopKModels: function (k)
##     id: branin
##     initialize: function (max.perf, max.ressources, prop.discard, s, B, id, par.set,
##     iteration: 1
##     max.perf: FALSE
##     max.ressources: NULL
##     models: list
##     n.configs: 1
##     par.set: NULL
##     printState: function ()
##     prop.discard: 3
##     r.config: 27
##     run: function ()
##     s: 1
##     sample.fun: NULL
##     step: function ()
##
## [[5]]
## <bracket>
##   Public:
##     B: 405
##     clone: function (deep = FALSE)
##     configurations: list
##     filterTopKModels: function (k)
##     getBudgetAllocation: function ()
##     getNumberOfModelsToSelect: function ()
##     getPerformances: function ()
##     getTopKModels: function (k)
##     id: branin
##     initialize: function (max.perf, max.ressources, prop.discard, s, B, id, par.set,
##     iteration: 0
##     max.perf: FALSE
##     max.ressources: NULL
##     models: list
##     n.configs: 1
##     par.set: NULL
##     printState: function ()
##     prop.discard: 3
##     r.config: 81
##     run: function ()
##     s: 0
##     sample.fun: NULL
##     step: function ()
```

We could inspect the best configuration of bracket 3:

```
hyperhyper[[3]]$models[[1]]$model
```

```
## [1] 3.226535 2.141058
```

Or the first three configurations of bracket 2:

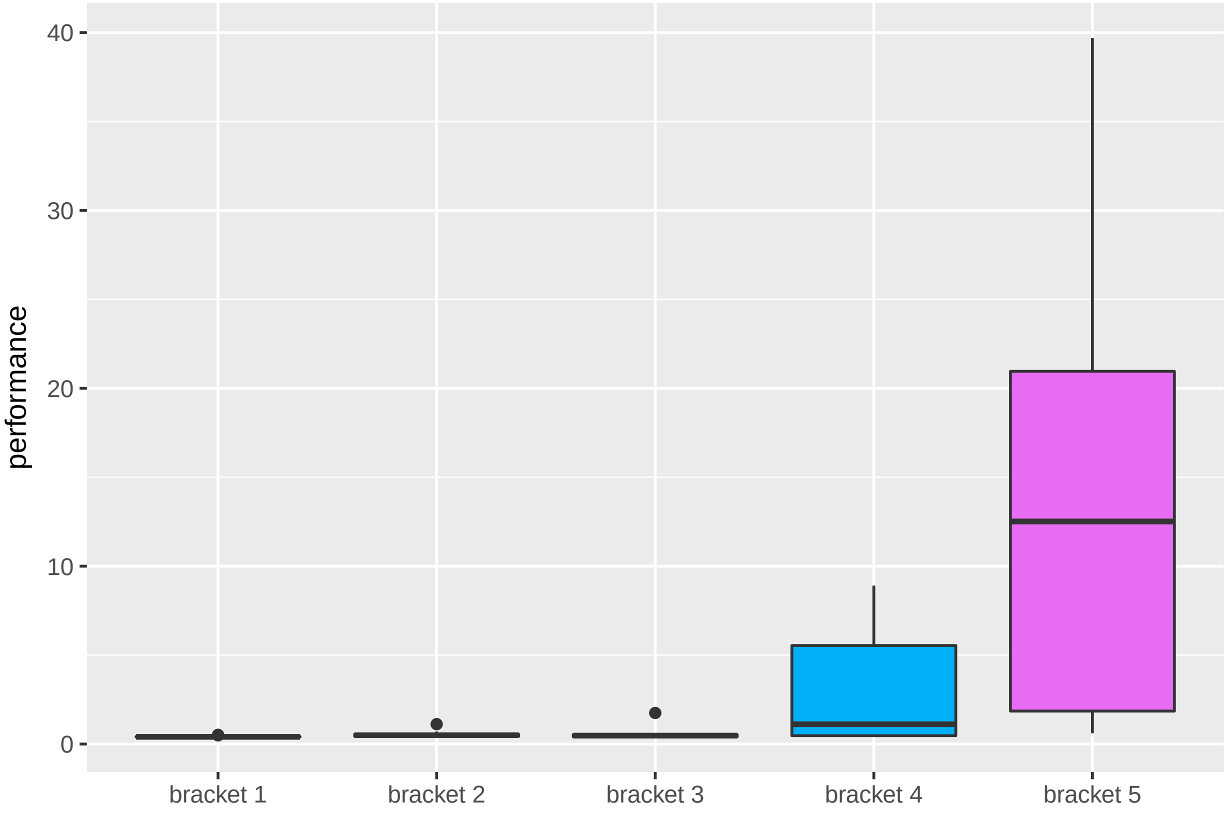
```
hyperhyper[[2]]$configurations[1:3]
```

```
## [[1]]
## [[1]]$x1
## [1] -2.599471
##
##
## [[2]]
## [[2]]$x1
## [1] 0.8387094
##
##
## [[3]]
## [[3]]$x1
## [1] 3.008917
```

Let us carry out a little benchmark to see if all brackets perform equally well here

Unsurprisingly, with hyperband standard configurations, bracket 5 does very bad on the branin problem (max.ressources = 81 and prop.discard = 3).

```
ggplot(stack(myBraninBenchmark), aes(x = ind, y = values, fill = ind)) +
  scale_x_discrete(labels=c("bracket 1","bracket 2","bracket 3","bracket 4", "bracket 5")) +
  theme(legend.position = "none") + labs(x = "", y = "performance") +
  geom_boxplot()
```


performance
bracket 1
bracket 2
bracket 3
bracket 4
bracket 5