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).

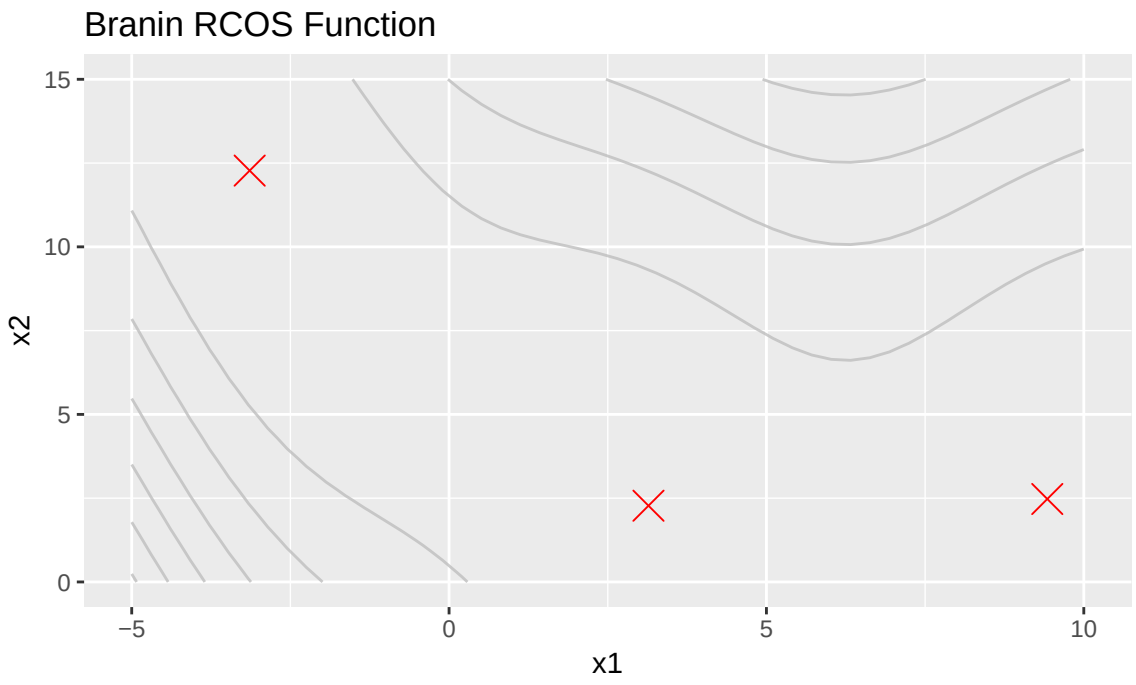
Simple example: optimize the branin function

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

```
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 of 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 four functions.

1: At first, we would like to have a function that samples random configurations

```
# par.set: the parameter space to sample from
# n.configs: the amount of configs to sample
sample.fun = function(par.set, n.configs) {
  runif(n = n.configs, -5, 10)
}
```

2: Now we also need a function to initialize each configuration as a model

```
#
config = runif(1, -5, 10.15)

# r: the initial budget for each config (for branin: not relevant)
# config: the configuration, e.g. for each x_1 the current value of x_2
init.fun = function(r, config) {
  runif(1, 0, 15)
}
```

3: Define a function to train each model

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.

```
# 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 = mod + rnorm(1, sd = 3)
    if(performance.fun(mod.new) < performance.fun(mod))
      mod = mod.new
  }
  return(mod)
}
```

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

```
# model: the model to evaluate
performance.fun = function(model) {
  problem(c(config, model))
}
```

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.ressources = 81,
  prop.discard = 3,
  bracket.winner = TRUE,
  id = "branin",
  par.set = NA,
  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: NULL
##     clone: function (deep = FALSE)
##     configurations: -1.33581830770709 6.7220373719465 -4.33212068979628 -2.4 ...
##     filterTopKModels: function (k)
##     getBudgetAllocation: function ()
##     getNumberOfModelsToSelect: function ()
##     getPerformances: function ()
##     getTopKModels: function (k)
##     id: branin
##     initialize: function (id, par.set, sample.fun, train.fun, performance.fun,
##     iteration: 4
##     max.perf: TRUE
##     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: NULL
##     clone: function (deep = FALSE)
##     configurations: 6.69505102792755 9.72414558753371 8.29847132670693 8.120 ...
##     filterTopKModels: function (k)
##     getBudgetAllocation: function ()
##     getNumberOfModelsToSelect: function ()
##     getPerformances: function ()
##     getTopKModels: function (k)
##     id: branin
##     initialize: function (id, par.set, sample.fun, train.fun, performance.fun,
##     iteration: 3
##     max.perf: TRUE
##     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: NULL
##     clone: function (deep = FALSE)
##     configurations: 0.158376034814864 1.4893687015865 0.958733503939584 6.01 ...
##     filterTopKModels: function (k)
##     getBudgetAllocation: function ()
##     getNumberOfModelsToSelect: function ()
##     getPerformances: function ()
##     getTopKModels: function (k)
##     id: branin
##     initialize: function (id, par.set, sample.fun, train.fun, performance.fun,
##     iteration: 2
##     max.perf: TRUE
##     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: NULL
##     clone: function (deep = FALSE)
##     configurations: 8.0504294030834 -3.28367307665758 -1.25577926286496 5.24 ...
##     filterTopKModels: function (k)
##     getBudgetAllocation: function ()
##     getNumberOfModelsToSelect: function ()
##     getPerformances: function ()
##     getTopKModels: function (k)
##     id: branin
##     initialize: function (id, par.set, sample.fun, train.fun, performance.fun,
##     iteration: 1
##     max.perf: TRUE
##     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: NULL
##     clone: function (deep = FALSE)
##     configurations: -3.13003169139847 5.24911931017414 -1.63770331768319 3.7 ...
##     filterTopKModels: function (k)
##     getBudgetAllocation: function ()
##     getNumberOfModelsToSelect: function ()
##     getPerformances: function ()
##     getTopKModels: function (k)
##     id: branin
##     initialize: function (id, par.set, sample.fun, train.fun, performance.fun,
##     iteration: 0
##     max.perf: TRUE
##     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 ()
```