Package 'ParBayesianOptimization'

October 22, 2019

Title Parallel Bayesian Optimization of Hyperparameters

Version 0.2.0

Description Fast, flexible framework for implementing Bayesian optimization of model hyperparameters according to the methods described in Snoek et al. <arXiv:1206.2944>. The package allows the user to run scoring function in parallel, save intermediary results, and tweak other aspects of the process to fully utilize the computing resources available to the user.

URL https://github.com/AnotherSamWilson/ParBayesianOptimization

BugReports https://github.com/AnotherSamWilson/ParBayesianOptimization/issues

Depends R (>= 3.4)

Imports data.table (>= 1.11.8), GauPro, stats, foreach, dbscan, lhs, ggplot2, plotly, crayon

Suggests knitr, rmarkdown, xgboost, doParallel

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

VignetteBuilder knitr

Maintainer Samuel Wilson <samwilson303@gmail.com>

NeedsCompilation no

Author Samuel Wilson [aut, cre]

Repository CRAN

Date/Publication 2019-10-22 04:30:02 UTC

R topics documented:

`BayesianOptimization` *Bayesian Optimization*

Description

Flexible Bayesian optimization of model hyperparameters.

Usage

```
BayesianOptimization(FUN, bounds, saveIntermediate = NULL,
  leftOff = NULL, parallel = FALSE, packages = NULL, export = NULL,
  initialize = TRUE, initGrid = NULL, initPoints = 0, bulkNew = 1,
  nIters = 0, kern = "Matern52", beta, acq = "ucb",
  stopImpatient = list(newAcq = "ucb", rounds = Inf), kappa = 2.576,
  eps = 0, gsPoints = 100, convThresh = 1e+07,
  minClusterUtility = NULL, noiseAdd = 0.25, plotProgress = TRUE,
  verbose = 1)
```

Arguments

`FUN`	the function to be maximized. This function should return a named list with at least 1 component. The first component must be named `Score` and should contain the metric to be maximized. You may return other named scalar elements that you wish to include in the final summary table.
`bounds`	named list of lower and upper bounds for each hyperparameter. The names of the list should be arguments passed to `FUN`. Use "L" suffix to indicate integer hyperparameters.
`saveIntermediate`	character filepath (including file name) that specifies the location to save intermediary results. This will save the ScoreDT data.table as an RDS. This RDS is saved after every iteration, and can be specified as the `leftOff` parameter so that you can continue a process where you left off.
`leftOff`	data.table containing parameter-Score pairs. If supplied, the process will `rbind` this table to the parameter-Score pairs obtained through initialization. This table should be obtained from either the file saved by `saveIntermediate`, or from the ScoreDT `data.table` returned by this function. WARNING: any parameters not within `bounds` will be removed before optimization takes place.
`parallel`	should the process run in parallel? If TRUE, several criteria must be met: • A parallel backend must be registered • `FUN` must be executable using only packages specified in `packages` (and base packages) • `FUN` must be executable using only the the objects specified in `export` • The function must be thread safe.
`packages`	character vector of the packages needed to run `FUN`.

export — character vector of object names needed to evaluate FUN.

initialize — should the process initialize a parameter-Score pair set? If FALSE, leftOff must be provided.

initGrid — user specified points to sample the scoring function, should be a data.frame or data.table with identical column names as bounds.

initPoints — Number of points to initialize the process with. Points are chosen with latin hypercube sampling within the bounds supplied.

bulkNew — integer that specifies the number of parameter combinations to sample at each optimization step. If minClusterUtility is NULL then noise is added to the acquisition optimum to obtain other sampling points. If running in parallel, good practice is to set bulkNew to some multiple of the number of cores you have designated for this process.

nIters — total number of parameter sets to be sampled, including initial set.

kern — a character that gets mapped to one of GauPro's GauPro_kernel_beta S6 classes. Determines the covariance function used in the gaussian process. Can be one of:

- "Gaussian"
- "Exponential"
- "Matern52"
- "Matern32"

beta — Depreciated. The kernel lengthscale parameter log10(theta). Passed to GauPro_kernel_beta specified in kern.

acq — acquisition function type to be used. Can be "ucb", "ei", "eips" or "poi".

- ucb Upper Confidence Bound
- ei Expected Improvement
- eips Expected Improvement Per Second
- poi Probability of Improvement

stopImpatient — a list containing rounds and newAcq, if acq = "eips" you can switch the acquisition function to newAcq after rounds parameter-score pairs are found.

kappa — tunable parameter kappa of the upper confidence bound. Adjusts exploitation/exploration. Increasing kappa will increase the importance that uncertainty (unexplored space) has, therefore incentivising exploration. This number represents the standard deviations above 0 of your upper confidence bound. Default is 2.56, which corresponds to the ~99th percentile.

eps — tunable parameter epsilon of ei, eips and poi. Adjusts exploitation/exploration. This value is added to y_max after the scaling, so should between -0.1 and 0.1. Increasing eps will make the "improvement" threshold for new points higher, therefore incentivising exploitation.

gsPoints — integer that specifies how many initial points to try when searching for the optimum of the acquisition function. Increase this for a higher chance to find global optimum, at the expense of more time.

convThresh — convergence threshold passed to factr when the optim function (L-BFGS-B) is called. Lower values will take longer to converge, but may be more accurate.

`minClusterUtility`
: number 0-1. Represents the minimum percentage of the optimal utility required for a less optimal local maximum to be included as a candidate parameter set in the next scoring function. If `NULL`, only the global optimum will be used as a candidate parameter set. If 0.5, only local optimums with 50 percent of the global optimum will be used.

`noiseAdd`
: specifies how much noise to add to acquisition optimums to obtain new parameter sets, if needed. New random draws are pulled from a shape(4,4) beta distribution centered at the optimal candidate parameter set with a range equal to `noiseAdd*(Upper Bound -Lower Bound)`

`plotProgress`
: Should the progress of the Bayesian optimization be printed? Top graph shows the score(s) obtained at each iteration. The bottom graph shows the optimal value of the acquisition function at each iteration. This is useful to display how much utility the Gaussian Process is actually assuming still exists. If your utility is approaching 0, then you can be confident you are close to an optimal parameter set.

`verbose`
: Whether or not to print progress to the console. If 0, nothing will be printed. If 1, progress will be printed. If 2, progress and information about new parameter-score pairs will be printed.

Value

A list containing details about the process:

`GPs`
: The last Gaussian process run on the parameter-score pairs

`GPe`
: If `acq = "eips"`, this contains the last Gaussian Process run on the parameter-elapsed time pairs

`progressPlot`
: a Plotly chart showing the evolution of the scores and utility discovered during the Bayesian optimization

`ScoreDT`
: A list of all parameter-score pairs, as well as extra columns from FUN. gpUtility is the acquisition function value at the time that parameter set was tested. acqOptimum is a boolean column that specifies whether the parameter set was an acquisition function optimum, or if it was obtained by applying noise to another optimum. Elapsed is the amount of time in seconds it took FUN to evaluate that parameter set.

`BestPars`
: The best parameter set at each iteration

References

Jasper Snoek, Hugo Larochelle, Ryan P. Adams (2012) *Practical Bayesian Optimization of Machine Learning Algorithms*

Examples

```
# Example 1 - Optimization of a continuous single parameter function
scoringFunction <- function(x) {
  a <- exp(-(2-x)^2)*1.5
  b <- exp(-(4-x)^2)*2
```

```
  c <- exp(-(6-x)^2)*1
  return(list(Score = a+b+c))
}

bounds <- list(x = c(0,8))

Results <- BayesianOptimization(
    FUN = scoringFunction
  , bounds = bounds
  , initPoints = 5
  , nIters = 8
  , gsPoints = 10
)

## Not run:
# Example 2 - Hyperparameter Tuning in xgboost
library("xgboost")

data(agaricus.train, package = "xgboost")

Folds <- list( Fold1 = as.integer(seq(1,nrow(agaricus.train$data),by = 3))
             , Fold2 = as.integer(seq(2,nrow(agaricus.train$data),by = 3))
             , Fold3 = as.integer(seq(3,nrow(agaricus.train$data),by = 3)))

scoringFunction <- function(max_depth, min_child_weight, subsample) {

  dtrain <- xgb.DMatrix(agaricus.train$data,label = agaricus.train$label)

  Pars <- list(
      booster = "gbtree"
    , eta = 0.01
    , max_depth = max_depth
    , min_child_weight = min_child_weight
    , subsample = subsample
    , objective = "binary:logistic"
    , eval_metric = "auc"
  )

  xgbcv <- xgb.cv(
       params = Pars
     , data = dtrain
     , nround = 100
     , folds = Folds
     , prediction = TRUE
     , showsd = TRUE
     , early_stopping_rounds = 5
     , maximize = TRUE
     , verbose = 0
  )

  return(list( Score = max(xgbcv$evaluation_log$test_auc_mean)
             , nrounds = xgbcv$best_iteration
  )
```

```
  )
}

bounds <- list(
    max_depth = c(2L, 10L)
  , min_child_weight = c(1, 100)
  , subsample = c(0.25, 1)
)

ScoreResult <- BayesianOptimization(
    FUN = scoringFunction
  , bounds = bounds
  , initPoints = 5
  , bulkNew = 1
  , nIters = 7
  , kern = "Matern52"
  , acq = "ei"
  , verbose = 1
  , parallel = FALSE
  , gsPoints = 50
)

## End(Not run)
```

Index