Package: GPopt (via r-universe)

July 22, 2026

Type Package

Title 'Bayesian' Optimization using Gaussian Process Regression and Other Surrogates (R Interface to Python's 'GPopt')

Version 0.10.2

Author T. Moudiki

Maintainer T. Moudiki <thierry.moudiki@gmail.com>

Description An R interface to the Python package 'GPopt' for 'Bayesian' Optimization using Gaussian Process Regression and Other Surrogates <https://github.com/Techtonique/GPopt>, for 'Bayesian' optimization of black-box (and machine learning hyperparameter tuning) objective functions using Gaussian Process Regression and other surrogate models. Ported to R using 'reticulate' and 'uv', following the same technique described in <https://thierrymoudiki.github.io/blog/2025/12/17/r/python/new-nnetsauce-R-uv>. Every function in this package thinly wraps the corresponding Python object: attribute and method access with '$' in R mirrors attribute and method access with '.' in Python.

License BSD_3_clause + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Imports reticulate

SystemRequirements Python (>= 3.9), the Python package GPopt (installed in a virtual environment, e.g. created with uv -- see README.md)

URL https://github.com/Techtonique/GPopt_r, https://github.com/Techtonique/GPopt

BugReports https://github.com/Techtonique/GPopt_r/issues

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3
Config/pak/sysreqs libpng-dev python3
Repository https://techtonique.r-universe.dev
Date/Publication 2026-07-22 08:13:30 UTC
RemoteUrl https://github.com/Techtonique/GPopt_r
RemoteRef HEAD
RemoteSha 7344108ca5e7fb3cfb94e0461265994717c3463e

Contents

`BOstopping` *BOstopping: Bayesian optimization with principled early stopping*

Description

R interface to Python's `GPopt.BOstopping`. A Gaussian-Process-based Bayesian optimizer with built-in early stopping diagnostics (based on e.g. Wasserstein-distance convergence of the posterior on a fixed test set), useful when function evaluations are expensive and you want the search to stop automatically once it has converged.

Usage

```
BOstopping(
  f,
  bounds,
  n_init = 5L,
  kappa = 1.96,
  early_stopping = TRUE,
  stop_patience = 20L,
  stop_threshold = 0.01,
  n_test_points = 100L,
  alpha = 1e-06,
  n_restarts_optimizer = 25L,
  seed = 123L,
  venv_path = "./venv"
)
```

Arguments

f	an R function taking a single numeric vector ‘x‘ (1-indexed, as usual in R) and returning a single numeric value to be minimized
bounds	a numeric matrix (or list of length-2 numeric vectors) with one row ‘c(lower, upper)‘ per dimension, e.g. ‘rbind(c(-5, 10), c(0, 15))‘
n_init	number of points in the initial design
kappa	exploration/exploitation trade-off parameter for the acquisition function
early_stopping	logical, whether to enable early stopping
stop_patience	number of iterations without sufficient improvement before stopping
stop_threshold	convergence threshold used by the early-stopping rule
n_test_points	number of points used to monitor convergence
alpha	numeric, GP regularization / noise parameter
n_restarts_optimizer	number of restarts of the GP's internal hyperparameter optimizer
seed	integer random seed
venv_path	path to the Python virtual environment created with ‘uv‘ (see the package README for setup instructions)

Value

A Python ‘GPopt.BOstopping‘ object, with ‘$optimize(n_iter = 100L)‘ as its main method (accessed with ‘$‘).

Examples

```
## Not run:
branin <- function(x) {
  x1 <- x[1]; x2 <- x[2]
  term1 <- (x2 - (5.1 * x1^2) / (4 * pi^2) + (5 * x1) / pi - 6)^2
  term2 <- 10 * (1 - 1 / (8 * pi)) * cos(x1)
  term1 + term2 + 10
}

opt <- BOstopping(
  f = branin,
  bounds = rbind(c(-5, 10), c(0, 15)),
  venv_path = "./venv"
)
result <- opt$optimize(n_iter = 100L)

## End(Not run)
```

`GeneralizationOpt` *GeneralizationOpt: diagnostics for the generalization gap of ML hyperparameters*

Description

R interface to Python's `GPopt.GeneralizationOpt`. Explores a hyperparameter space with Sobol sequences, fits a surrogate model to the generalization gap (train vs. test performance), and lets you compute Sobol sensitivity indices and run diagnostics – useful to understand *which* hyperparameters drive over/under-fitting, beyond just tuning for best validation score.

Usage

```
GeneralizationOpt(hyperparams, random_state = 42L, venv_path = "./venv")
```

Arguments

`hyperparams`	a named list describing the hyperparameter space, in the form 'list(param_name = c(min_val, max_val, log_scale))', e.g. 'list(nu = c(0.001, 10, TRUE), lam = c(1e-4, 1e4, TRUE))'
`random_state`	integer random seed
`venv_path`	path to the Python virtual environment created with 'uv' (see the package README for setup instructions)

Value

A Python 'GPopt.GeneralizationOpt' object. Useful members (accessed with '$'):

- '$generate_sobol_samples(n = ...)'
- '$evaluate_configurations(...)'
- '$fit_surrogate(surrogate = ..., target_gap = "gap_abs")'
- '$compute_sobol_indices(...)'
- '$optimize_acquisition(...)'
- '$run_diagnostics(...)'

Examples

```
## Not run:
gopt <- GeneralizationOpt(
  hyperparams = list(
    nu  = c(0.001, 10, TRUE),
    lam = c(1e-4, 1e4, TRUE)
  ),
  venv_path = "./venv"
)
samples <- gopt$generate_sobol_samples(n = 64L)

## End(Not run)
```

GenericSurrogate *GenericSurrogate: wrap any scikit-learn-compatible regressor as a GPOpt surrogate*

Description

R interface to Python's `GPopt.GenericSurrogate`. Turns an arbitrary scikit-learn-compatible regressor (obtained e.g. with [get_sklearn()]) into a conformalized and/or Bayesian surrogate model, usable as the 'surrogate_obj' argument of [GPOpt()] instead of the default Gaussian Process Regression surrogate.

Usage

```
GenericSurrogate(
  base_model,
  conformal = TRUE,
  bayesian = FALSE,
  venv_path = "./venv"
)
```

Arguments

`base_model`	a Python scikit-learn-compatible regressor *instance* (not a class), e.g. 'get_sklearn(venv_path)$ensembl
`conformal`	logical, whether to conformalize the surrogate's predictions (for calibrated uncertainty quantification)
`bayesian`	logical, whether to use a Bayesian variant of the surrogate
`venv_path`	path to the Python virtual environment created with 'uv' (see the package README for setup instructions)

Value

A Python 'GPopt.GenericSurrogate' object, with scikit-learn-style members accessible with '$': '$fit(X, y)', '$predict(X)', '$score(X, y)', '$get_params()', '$set_params(...)'.

Examples

```
## Not run:
sklearn <- get_sklearn(venv_path = "./venv")
base_model <- sklearn$ensemble$ExtraTreesRegressor()
surrogate <- GenericSurrogate(base_model, conformal = TRUE, venv_path = "./venv")

opt <- GPOpt(
  lower_bound = c(-5, 0),
  upper_bound = c(10, 15),
  objective_func = function(x) sum(x^2),
  surrogate_obj = surrogate,
  venv_path = "./venv"
```

```
)

## End(Not run)
```

`get_GPopt` *Get the Python* `GPopt` *module*

Description

Thin wrapper around `reticulate::import("GPopt")`, after switching reticulate to the virtual environment at `venv_path`. Useful if you want to access parts of the Python API that don't (yet) have a dedicated R constructor in this package – e.g. `get_GPopt()$config` or `get_GPopt()$genericcv`.

Usage

```
get_GPopt(venv_path = "./venv")
```

Arguments

`venv_path`	path to the Python virtual environment created with 'uv' (see the package README for setup instructions)

Value

the Python module 'GPopt', usable with '$' the way you would use '.' in Python (e.g. 'get_GPopt()$GPOpt(...)')

Examples

```
## Not run:
gp <- get_GPopt(venv_path = "./venv")
gp$GPOpt

## End(Not run)
```

`get_numpy` *Get the Python* `numpy` *module*

Description

Get the Python `numpy` module

Usage

```
get_numpy(venv_path = "./venv")
```

Arguments

venv_path	path to the Python virtual environment created with ‘uv‘ (see the package README for setup instructions)

Value

the Python module ‘numpy‘

---

get_sklearn *Get the Python* scikit-learn *module*

Description

Handy when building objective functions for [GPOpt()] or estimator classes/instances for [MLOptimizer()] and [GenericSurrogate()], without leaving R.

Usage

```
get_sklearn(venv_path = "./venv")
```

Arguments

venv_path	path to the Python virtual environment created with ‘uv‘ (see the package README for setup instructions)

Value

the Python module ‘sklearn‘

Examples

```
## Not run:
sklearn <- get_sklearn(venv_path = "./venv")
rf <- sklearn$ensemble$RandomForestClassifier

## End(Not run)
```

GPOpt *GPOpt: Bayesian optimization using Gaussian Process Regression*

Description

R interface to Python's `GPopt.GPOpt`, the main class of the `GPopt` package for minimizing a black-box, expensive-to-evaluate objective function over a box-constrained search space.

Usage

```
GPOpt(
  lower_bound,
  upper_bound,
  objective_func = NULL,
  params_names = NULL,
  surrogate_obj = NULL,
  x_init = NULL,
  y_init = NULL,
  n_init = 10L,
  n_choices = 100000L,
  n_iter = 190L,
  alpha = 1e-06,
  n_restarts_optimizer = 25L,
  seed = 123L,
  save = NULL,
  n_jobs = 1L,
  acquisition = c("ei", "ucb"),
  method = "bayesian",
  min_value = NULL,
  per_second = FALSE,
  log_scale = FALSE,
  venv_path = "./venv",
  ...
)
```

Arguments

`lower_bound`	numeric vector, lower bound for the parameters to optimize
`upper_bound`	numeric vector, upper bound for the parameters to optimize
`objective_func`	an R function taking a single numeric vector 'x' (1-indexed, as usual in R – e.g. 'x[1]', 'x[2]', ...) and returning a single numeric value to be minimized. Can be left 'NULL', e.g. when the optimizer state will be '$load()'-ed from disk.
`params_names`	optional character vector, names for the parameters (makes results easier to map back onto keyword arguments of an ML model)

surrogate_obj	optional surrogate model object (e.g. the result of [GenericSurrogate()], or any Python scikit-learn-compatible regressor obtained via [get_sklearn()]) used instead of the default Gaussian Process Regression surrogate
x_init, y_init	optional numeric matrix/vector with an existing initial design (‘x_init‘) and corresponding objective values (‘y_init‘)
n_init	number of points in the initial (space-filling) design
n_choices	number of candidate points used when searching for the next evaluation point
n_iter	number of iterations of the optimizer
alpha	numeric, GP regularization / noise parameter
n_restarts_optimizer	number of restarts of the GP’s internal hyperparameter optimizer
seed	integer random seed
save	optional path (character) to a shelve file used to save and resume the optimization (see the "save and resume" example in the original Python package’s documentation)
n_jobs	number of jobs to run in parallel
acquisition	acquisition function, one of ‘"ei"‘ (expected improvement) or ‘"ucb"‘ (upper confidence bound)
method	‘"bayesian"‘ (Gaussian Process surrogate, the default) or ‘"mc"‘/other methods implemented by the Python package
min_value	optional known minimum value of the objective, used for early stopping diagnostics
per_second	logical, whether the objective function also returns timing information (see Python package docs)
log_scale	logical, whether to search on a log scale
venv_path	path to the Python virtual environment created with ‘uv‘ (see the package README for setup instructions)
...	further named arguments forwarded as-is to ‘GPopt.GPOpt(...)‘ on the Python side

Details

This function returns the underlying Python object itself (a `GPopt.GPOpt` instance, wrapped by `reticulate`). As with the ‘nnetsauce‘ port, the general rule is: object accesses with ‘.‘s in Python are replaced by ‘$‘s in R. So after building ‘opt <- GPOpt(...)‘, call ‘opt$optimize()‘, read ‘opt$x_min‘, ‘opt$y_min‘, ‘opt$n_iter‘, ‘opt$max_ei‘, etc., exactly like you would in Python with ‘opt.optimize()‘, ‘opt.x_min‘, and so on.

Value

A Python ‘GPopt.GPOpt‘ object. Useful members (accessed with ‘$‘):

- ‘$optimize(verbose = 1L, n_more_iter = NULL, abs_tol = NULL, ucb_tol = NULL, ...)‘ runs (or resumes) the optimization loop and returns a ‘DescribeResult‘ (an R list with elements ‘best_params‘/‘best_score‘ once it crosses over to R)

- '$lazyoptimize(...)' tries several surrogate models automatically
- '$x_min', '$y_min' current best parameters/objective value
- '$n_iter' current number of iterations actually run
- '$max_ei' history of maximum expected improvement per iteration
- '$save(...)'/'$load(path = ...)'/'$close_shelve()' persistence helpers

Examples

```
## Not run:
# Branin function, minimized over [-5, 10] x [0, 15]
branin <- function(x) {
  x1 <- x[1]; x2 <- x[2]
  term1 <- (x2 - (5.1 * x1^2) / (4 * pi^2) + (5 * x1) / pi - 6)^2
  term2 <- 10 * (1 - 1 / (8 * pi)) * cos(x1)
  term1 + term2 + 10
}

opt <- GPOpt(
  lower_bound = c(-5, 0),
  upper_bound = c(10, 15),
  objective_func = branin,
  n_init = 10,
  n_iter = 90,
  venv_path = "./venv"
)
res <- opt$optimize(verbose = 2L)
print(opt$x_min)
print(opt$y_min)


## End(Not run)
```

`MLOptimizer` *MLOptimizer: automated hyperparameter tuning for scikit-learn-compatible estimators*

Description

R interface to Python's `GPopt.MLOptimizer`. A convenience layer on top of [GPOpt()] specialized for cross-validated tuning of a scikit-learn estimator class: you provide training data, an estimator *class* (not an instance), and a parameter configuration (bounds/transforms/dtypes), and '$optimize()' runs the cross-validated Bayesian search for you.

Usage

```
MLOptimizer(
  scoring = "accuracy",
  cv = 5L,
```

```
  n_jobs = NULL,
  n_init = 10L,
  n_iter = 90L,
  seed = 3137L,
  venv_path = "./venv"
)
```

Arguments

scoring	scikit-learn scoring string, e.g. ‘"accuracy"‘, ‘"r2"‘, ‘"neg_mean_squared_error"‘
cv	number of cross-validation folds
n_jobs	number of jobs to run in parallel during cross-validation (‘NULL‘ uses scikit-learn’s default)
n_init	number of points in the initial design
n_iter	number of iterations of the optimizer
seed	integer random seed
venv_path	path to the Python virtual environment created with ‘uv‘ (see the package README for setup instructions)

Value

A Python ‘GPopt.MLOptimizer‘ object. Useful members (accessed with ‘$‘):

- ‘$optimize(X_train, y_train, estimator_class, param_config, verbose = 2L)‘
- ‘$get_best_parameters(apply_transforms = TRUE)‘
- ‘$get_best_score()‘
- ‘$create_optimized_estimator()‘
- ‘$fit_optimized_estimator(X_train, y_train)‘

Examples

```
## Not run:
sklearn <- get_sklearn(venv_path = "./venv")
RandomForestClassifier <- sklearn$ensemble$RandomForestClassifier

X <- as.matrix(iris[, 1:4])
y <- as.integer(iris$Species) - 1L

mlopt <- MLOptimizer(scoring = "accuracy", cv = 5L, n_iter = 90L, venv_path = "./venv")

param_config <- list(
  n_estimators = list(bounds = c(10, 300), dtype = "int"),
  max_depth    = list(bounds = c(1, 20), dtype = "int")
)

mlopt$optimize(
  X_train = X, y_train = y,
  estimator_class = RandomForestClassifier(),
```

```
  param_config = param_config,
  verbose = 2L
)
mlopt$get_best_parameters()
mlopt$get_best_score()

## End(Not run)
```

Index