Package ‘mlr3mbo’

March 1, 2024

Type Package
Title Flexible Bayesian Optimization
Version 0.2.2
Description A modern and flexible approach to Bayesian Optimization / Model
Based Optimization building on the ‘bbotk’ package. ‘mlr3mbo’ is a toolbox
providing both ready-to-use optimization algorithms as well as their fundamental
building blocks allowing for straightforward implementation of custom
algorithms. Single- and multi-objective optimization is supported as well as
mixed continuous, categorical and conditional search spaces. Moreover, using
‘mlr3mbo’ for hyperparameter optimization of machine learning models within the
‘mlr3’ ecosystem is straightforward via ‘mlr3tuning’. Examples of ready-to-use
optimization algorithms include Efficient Global Optimization by Jones et al.
<doi:10.1109/TEVC.2005.851274> and SMS-EGO by Ponweiser et al. (2008)
<doi:10.1007/978-3-540-87700-4_78>.

License LGPL-3
BugReports https://github.com/mlr-org/mlr3mbo/issues
Depends R (>= 3.1.0)
Imports bbotk (>= 0.5.4), checkmate (>= 2.0.0), data.table, lgr (>=
0.3.4), mlr3 (>= 0.14.0), mlr3misc (>= 0.11.0), mlr3tuning (>=
0.14.0), paradox (>= 0.10.0), spacefillr, R6 (>= 2.4.1)
Suggests DiceKriging, emoa, fastGHQuad, knitr, lhs, mlr3learners (>=
0.5.4), mlr3pipelines (>= 0.4.2), nloptr, ranger, rgenoud,
markdown, rpart, stringi, testthat (>= 3.0.0),

ByteCompile no
Encoding UTF-8
Config/testthat/edition 3
Config/testthat/parallel false
NeedsCompilation yes
RoxygenNote 7.3.1
Collate 'mlr_acqfunctions.R' 'AcqFunction.R' 'AcqFunctionAEI.R'
   'AcqFunctionCB.R' 'AcqFunctionEHVI.R' 'AcqFunctionEVI.R'
   'AcqFunctionEI.R' 'AcqFunctionEIPS.R' 'AcqFunctionMean.R'
   'AcqFunctionPI.R' 'AcqFunctionSmsEgo.R'
   'AcqOptimizer.R' 'aaa.R' 'OptimizerMbo.R'
   'mlr_result_assigners.R' 'ResultAssigner.R'
   'ResultAssignerArchive.R' 'ResultAssignerSurrogate.R'
   'Surrogate.R' 'SurrogateLearner.R'
   'SurrogateLearnerCollection.R' 'TunerMbo.R'
   'mlr_loop_functions.R' 'bayesopt_ego.R' 'bayesopt_emo.R'
   'bayesopt_mpcl.R' 'bayesopt_parego.R' 'bayesopt_smsego.R'
   'bibentries.R' 'helper.R' 'loop_function.R' 'mbo_defaults.R'
   'sugar.R' 'zzz.R'

VignetteBuilder knitr

Author Lennart Schneider [cre, aut] (https://orcid.org/0000-0003-4152-5308),
   Jakob Richter [aut] (https://orcid.org/0000-0003-4481-5554),
   Marc Becker [aut] (https://orcid.org/0000-0002-8115-0400),
   Michel Lang [aut] (https://orcid.org/0000-0001-9754-0393),
   Bernd Bischl [aut] (https://orcid.org/0000-0001-6002-6980),
   Florian Pfisterer [aut] (https://orcid.org/0000-0001-8867-762X),
   Martin Binder [aut],
   Sebastian Fischer [aut] (https://orcid.org/0000-0002-9609-3197),
   Michael H. Buselli [cph],
   Wessel Dankers [cph],
   Carlos Fonseca [cph],
   Manuel Lopez-Ibanez [cph],
   Luis Paquete [cph]

Maintainer Lennart Schneider <lennart.sch@web.de>

Repository CRAN

Date/Publication 2024-03-01 19:40:02 UTC

R topics documented:

  mlr3mbo-package .................................................. 3
  acqf ............................................................ 4
  AcqFunction ..................................................... 5
  acqo ............................................................ 7
  AcqOptimizer .................................................... 8
  default_acqfunction ............................................. 11
  default_acqoptimizer ........................................... 11
  default_gp ....................................................... 12
  default_loop_function .......................................... 13
  default_result_assigner ........................................ 13
  default_rf ....................................................... 14
  default_surrogate ............................................... 14
  loop_function ................................................... 16
Description

A modern and flexible approach to Bayesian Optimization / Model Based Optimization building on the 'bbotk' package. 'mlr3mbo' is a toolbox providing both ready-to-use optimization algorithms as well as their fundamental building blocks allowing for straightforward implementation of custom algorithms. Single- and multi-objective optimization is supported as well as mixed continuous, categorical and conditional search spaces. Moreover, using 'mlr3mbo' for hyperparameter optimization of machine learning models within the 'mlr3' ecosystem is straightforward via 'mlr3tuning'. Examples of ready-to-use optimization algorithms include Efficient Global Optimization by Jones et al. (1998) doi:10.1023/A:1008306431147, ParEGO by Knowles (2006) doi:10.1109/TEVC.2005.851274 and SMS-EGO by Ponweiser et al. (2008) doi:10.1007/9783540-877004_78.
Author(s)

Maintainer: Lennart Schneider <lennart.sch@web.de> (ORCID)

Authors:

- Jakob Richter <jakob1richter@gmail.com> (ORCID)
- Marc Becker <marcbecker@posteo.de> (ORCID)
- Michel Lang <michellang@gmail.com> (ORCID)
- Bernd Bischl <bernd_bischl@gmx.net> (ORCID)
- Florian Pfisterer <pfistererf@googlemail.com> (ORCID)
- Martin Binder <mlr.developer@mb706.com>
- Sebastian Fischer <sebf.fischer@gmail.com> (ORCID)

Other contributors:

- Michael H. Buselli [copyright holder]
- Wessel Dankers [copyright holder]
- Carlos Fonseca [copyright holder]
- Manuel Lopez-Ibanez [copyright holder]
- Luis Paquete [copyright holder]

See Also

Useful links:

- https://mlr3mbo.mlr-org.com
- https://github.com/mlr-org/mlr3mbo
- Report bugs at https://github.com/mlr-org/mlr3mbo/issues

acqf

Syntactic Sugar Acquisition Function Construction

Description

This function complements mlr_acqfunctions with functions in the spirit of mlr_sugar from mlr3.

Usage

acqf(.key, ...)

Arguments

.key (character(1))
Key passed to the respective dictionary to retrieve the object.

... (named list())
Named arguments passed to the constructor, to be set as parameters in the paradox::ParamSet, or to be set as public field. See mlr3misc::dictionary_sugar_get() for more details.
**AcqFunction**

**Value**

AcqFunction

**Examples**

acqf("ei")

---

<table>
<thead>
<tr>
<th>AcqFunction</th>
<th>Acquisition Function Base Class</th>
</tr>
</thead>
</table>

**Description**

Abstract acquisition function class.

Based on the predictions of a Surrogate, the acquisition function encodes the preference to evaluate a new point.

**Super class**

`bbotk::Objective` -> `AcqFunction`

**Active bindings**

direction ("same" | "minimize" | "maximize")

Optimization direction of the acquisition function relative to the direction of the objective function of the `bbotk::OptimInstance`. Must be "same", "minimize", or "maximize".

surrogate_max_to_min (-1 | 1)

Multiplicative factor to correct for minimization or maximization of the acquisition function.

label (character(1))

Label for this object.

man (character(1))

String in the format [pkg]:[topic] pointing to a manual page for this object.

archive (bbotk::Archive)

Points to the `bbotk::Archive` of the surrogate.

fun (function)

Points to the private acquisition function to be implemented by subclasses.

surrogate (Surrogate)

Surrogate.

requires_predict_type_se (logical(1))

Whether the acquisition function requires the surrogate to have "se" as $predict_type.

packages (character())

Set of required packages.
Methods

Public methods:

• `AcqFunction$new()`
• `AcqFunction$update()`
• `AcqFunction$eval_many()`
• `AcqFunction$eval_dt()`
• `AcqFunction$clone()`

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

Usage:
```r
AcqFunction$new(
  id,  # character(1)
  constants = ParamSet$new(),  # paradox::ParamSet
  surrogate,  # NULL | Surrogate
  requires_predict_type_se,  # logical(1)
  direction,  # "same" | "minimize" | "maximize"
  packages = NULL,  # character()
  label = NA_character_,  # character(1)
  man = NA_character_  # character(1)
)
```

Arguments:
- `id` (character(1)).
- `constants` (paradox::ParamSet). Changeable constants or parameters.
- `surrogate` (NULL | Surrogate). Surrogate whose predictions are used in the acquisition function.
- `requires_predict_type_se` (logical(1))
  Whether the acquisition function requires the surrogate to have "se" as $predict_type.
- `direction` ("same" | "minimize" | "maximize"). Optimization direction of the acquisition function relative to the direction of the objective function of the bbotk::OptimInstance. Must be "same", "minimize", or "maximize".
- `packages` (character())
  Set of required packages. A warning is signaled prior to construction if at least one of the packages is not installed, but loaded (not attached) later on-demand via requireNamespace().
- `label` (character(1))
  Label for this object.
- `man` (character(1))
  String in the format [pkg]::[topic] pointing to a manual page for this object.

Method `update()`: Update the acquisition function.

Can be implemented by subclasses.

Usage:
```r
AcqFunction$update()
```

Method `eval_many()`: Evaluates multiple input values on the objective function.
Usage:
AcqFunction$eval_many(xss)

Arguments:
xss (list())
    A list of lists that contains multiple x values, e.g. list(list(x1 = 1, x2 = 2), list(x1 = 3, x2 = 4)).

Returns: data.table::data.table() that contains one y-column for single-objective functions and multiple y-columns for multi-objective functions, e.g. data.table(y = 1:2) or data.table(y1 = 1:2, y2 = 3:4).

Method eval_dt(): Evaluates multiple input values on the objective function

Usage:
AcqFunction$eval_dt(xdt)

Arguments:
xdt (data.table::data.table())
    One point per row, e.g. data.table(x1 = c(1, 3), x2 = c(2, 4)).

Returns: data.table::data.table() that contains one y-column for single-objective functions and multiple y-columns for multi-objective functions, e.g. data.table(y = 1:2) or data.table(y1 = 1:2, y2 = 3:4).

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

Usage:
AcqFunction$clone(deep = FALSE)

Arguments:
    deep Whether to make a deep clone.

See Also

Other Acquisition Function: mlr_acqfunctions, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego

Description

This function allows to construct an AcqOptimizer in the spirit of mlr_sugar from mlr3.

Usage

acqo(optimizer, terminator, acq_function = NULL, ...)
AcqOptimizer

Arguments

- **optimizer** *(bbotk::Optimizer)*  
  *bbotk::Optimizer* that is to be used.
- **terminator** *(bbotk::Terminator)*  
  *bbotk::Terminator* that is to be used.
- **acq_function** *(NULL | AcqFunction)*  
  *AcqFunction* that is to be used. Can also be NULL.
- **...** *(named list())*
  Named arguments passed to the constructor, to be set as parameters in the *paradox::ParamSet*.

Value

*AcqOptimizer*

Examples

```r
library(bbotk)
acqo(opt("random_search"), trm("evals"), catch_errors = FALSE)
```

---

**AcqOptimizer**  
*Acquisition Function Optimizer*

Description

Optimizer for *AcqFunctions* which performs the acquisition function optimization. Wraps an *bbotk::Optimizer* and *bbotk::Terminator*.

Parameters

- **n_candidates** integer(1)
  Number of candidate points to propose. Note that this does not affect how the acquisition function itself is calculated (e.g., setting `n_candidates > 1` will not result in computing the q- or multi-Expected Improvement) but rather the top `n-candidates` are selected from the *bbotk::Archive* of the acquisition function *bbotk::OptimInstance*. Note that setting `n_candidates > 1` is usually not a sensible idea but it is still supported for experimental reasons. Default is 1.

- **logging_level** character(1)
  Logging level during the acquisition function optimization. Can be "fatal", "error", "warn", "info", "debug" or "trace". Default is "warn", i.e., only warnings are logged.

- **warmstart** logical(1)
  Should the acquisition function optimization be warm-started by evaluating the best point(s) present in the *bbotk::Archive* of the actual *bbotk::OptimInstance*? This is sensible when using a population based acquisition function optimizer, e.g., local search or mutation. Default is FALSE.
AcqOptimizer

warmstart_size integer(1) | "all"
Number of best points selected from the bbotk::Archive that are to be used for warm starting.
Can also be "all" to use all available points. Only relevant if warmstart = TRUE. Default is 1.

skip_already_evaluated logical(1)
It can happen that the candidate resulting of the acquisition function optimization was already evaluated in a previous iteration. Should this candidate proposal be ignored and the next best point be selected as a candidate? Default is TRUE.

catch_errors logical(1)
Should errors during the acquisition function optimization be caught and propagated to the loop_function which can then handle the failed acquisition function optimization appropriately by, e.g., proposing a randomly sampled point for evaluation? Default is TRUE.

Public fields

optimizer (bbotk::Optimizer).
terminator (bbotk::Terminator).
acq_function (AcqFunction).

Active bindings

print_id (character)
Id used when printing.

param_set (paradox::ParamSet)
Set of hyperparameters.

Methods

Public methods:

• AcqOptimizer$new()
• AcqOptimizer$format()
• AcqOptimizer$sprint()
• AcqOptimizer$optimize()
• AcqOptimizer$clone()

Method new(): Creates a new instance of this R6 class.
Usage:
AcqOptimizer$new(optimizer, terminator, acq_function = NULL)

Arguments:
optimizer (bbotk::Optimizer).
terminator (bbotk::Terminator).
acq_function (NULL | AcqFunction).

Method format(): Helper for print outputs.
Usage:
AcqOptimizer$format()
Method `print()`: Print method.

Usage:
```
AcqOptimizer/print()
```

Returns: (character()).

Method `optimize()`: Optimize the acquisition function.

Usage:
```
AcqOptimizer/optimize()
```

Returns: `data.table::data.table()` with 1 row per optimum and x as columns.

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

Usage:
```
AcqOptimizer/clone(deep = FALSE)
```

Arguments:
```
depth  Whether to make a deep clone.
```

Examples

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {
  library(bbotk)
  library(paradox)
  library(mlr3learners)
  library(data.table)

  fun = function(xs) {
    list(y = xs$x ^ 2)
  }
  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceSingleCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))

  instance$eval_batch(data.table(x = c(-6, -5, 3, 9)))

  learner = default_gp()
  surrogate = srlrn(learner, archive = instance$archive)
  acq_function = acqf("ei", surrogate = surrogate)
  acq_function$surrogate$update()
  acq_function$update()

  acq_optimizer = acqo(
```
**default_acqfunction**  

```r
optimizer = opt("random_search", batch_size = 1000),
terminator = trm("evals", n_evals = 1000),
acq_function = acq_function)

acq_optimizer$optimize()
```

---

**default_acqfunction**  

*Default Acquisition Function*

**Description**

Chooses a default acquisition function, i.e. the criterion used to propose future points. For single-objective optimization, defaults to `mlr_acqfunctions_ei`. For multi-objective optimization, defaults to `mlr_acqfunctions_smsego`.

**Usage**

```r
default_acqfunction(instance)
```

**Arguments**

- `instance` *(bbotk::OptimInstance)*

**Value**

AcqFunction

**See Also**

Other `mbo_defaults`: `default_acqoptimizer()`, `default_gp()`, `default_loop_function()`, `default_result_assigner()`, `default_rf()`, `default_surrogate()`, `mbo_defaults`

---

**default_acqoptimizer**  

*Default Acquisition Function Optimizer*

**Description**

Chooses a default acquisition function optimizer. Defaults to wrapping `bbotk::OptimizerRandomSearch` allowing 10000 function evaluations (with a batch size of 1000) via a `bbotk::TerminatorEvals`.

**Usage**

```r
default_acqoptimizer(acq_function)
```
**Arguments**

acq_function (AcqFunction).

**Value**

AcqOptimizer

**See Also**

Other mbo_defaults: default_acqfunction(), default_gp(), default_loop_function(), default_result_assigner(), default_rf(), default_surrogate(), mbo_defaults

---

**Description**

This is a helper function that constructs a default Gaussian Process mlr3::LearnerRegr which is for example used in default_surrogate.

Constructs a Kriging learner ""regr.km"" with kernel ""matern5_2"". If noisy = FALSE (default) a small nugget effect is added nugget.stability = 10^-8 to increase numerical stability to hopefully prevent crashes of DiceKriging. If noisy = TRUE the nugget effect will be estimated with nugget.estim = TRUE. If noisy = TRUE jitter is set to TRUE to circumvent a problem with DiceKriging where already trained input values produce the exact trained output. In general, instead of the default "BFGS" optimization method we use rgenoud ("gen"), which is a hybrid algorithm, to combine global search based on genetic algorithms and local search based on gradients. This may improve the model fit and will less frequently produce a constant model prediction.

**Usage**

default_gp(noisy = FALSE)

**Arguments**

noisy (logical(1))

Whether the learner will be used in a noisy objective function scenario. See above.

**Value**

mlr3::LearnerRegr

**See Also**

Other mbo_defaults: default_acqfunction(), default_acqoptimizer(), default_loop_function(), default_result_assigner(), default_rf(), default_surrogate(), mbo_defaults
**default_loop_function**  
*Default Loop Function*

**Description**
Chooses a default `loop_function`, i.e. the Bayesian Optimization flavor to be used for optimization. For single-objective optimization, defaults to `bayesopt_ego`. For multi-objective optimization, defaults to `bayesopt_smsego`.

**Usage**
```
default_loop_function(instance)
```

**Arguments**
- `instance` *(bbotk::OptimInstance)*  
  An object that inherits from `bbotk::OptimInstance`.

**Value**
`loop_function`

**See Also**
- Other `mbo_defaults`: `default_acqfunction()`, `default_acqoptimizer()`, `default_gp()`, `default_result_assigner()`, `default_rf()`, `default_surrogate()`, `mbo_defaults`

**default_result_assigner**  
*Default Result Assigner*

**Description**
Chooses a default result assigner. Defaults to `ResultAssignerArchive`.

**Usage**
```
default_result_assigner(instance)
```

**Arguments**
- `instance` *(bbotk::OptimInstance)*  
  An object that inherits from `bbotk::OptimInstance`.

**Value**
`ResultAssigner`
default_surrogate

See Also

Other mbo_defaults: default_acqfunction(), default_acqoptimizer(), default_gp(), default_loop_function(), default_rf(), default_surrogate(), mbo_defaults

default_rf

Default Random Forest

Description

This is a helper function that constructs a default random forest mlr3::LearnerRegr which is for example used in default_surrogate.

Constructs a ranger learner ""regr.ranger"" with num.trees = 100, keep.inbag = TRUE and se.method = "jack".

Usage

default_rf(noisy = FALSE)

Arguments

noisy (logical(1))
Whether the learner will be used in a noisy objective function scenario. Currently has no effect.

Value

mlr3::LearnerRegr

See Also

Other mbo defaults: default_acqfunction(), default_acqoptimizer(), default_gp(), default_loop_function(), default_result_assigner(), default_surrogate(), mbo_defaults

default_surrogate

Default Surrogate
**default_surrogate**

**Description**

This is a helper function that constructs a default Surrogate based on properties of the bbotk::OptimInstance.

For numeric-only (including integers) parameter spaces without any dependencies a Gaussian Process is constricted via default_gp(). For mixed numeric-categorical parameter spaces, or spaces with conditional parameters a random forest is constructed via default_rf().

In any case, learners are encapsulated using ""evaluate"", and a fallback learner is set, in cases where the surrogate learner errors. Currently, the following learner is used as a fallback: lrn("regr.ranger", num.trees = 10L, keep.inbag = TRUE, se.method = "jack").

If additionally dependencies are present in the parameter space, inactive conditional parameters are represented by missing NA values in the training design data. We simply handle those with an imputation method, added to the random forest, more concretely we use po("imputesample") (for logicals) and po("imputeoor") (for anything else) from package mlr3pipelines. Characters are always encoded as factors via po("colapply"). Out of range imputation makes sense for tree-based methods and is usually hard to beat, see Ding et al. (2010). In the case of dependencies, the following learner is used as a fallback: lrn("regr.featureless").

If the instance is of class bbotk::OptimInstanceSingleCrit the learner is wrapped as a SurrogateLearner.

If the instance is of class bbotk::OptimInstanceMultiCrit multiple deep clones of the learner are wrapped as a SurrogateLearnerCollection.

**Usage**

default_surrogate(instance, learner = NULL, n_learner = NULL)

**Arguments**

- **instance** (bbotk::OptimInstance)
  An object that inherits from bbotk::OptimInstance.

- **learner** (NULL | mlr3::Learner). If specified, this learner will be used instead of the defaults described above.

- **n_learner** (NULL | integer(1)). Number of learners to be considered in the construction of the SurrogateLearner or SurrogateLearnerCollection. If not specified will be based on the number of objectives as stated by the instance.

**Value**

Surrogate

**References**

See Also

Other mbo_defaults: default_acqfunction(), default_acqoptimizer(), default_gp(), default_loop_function(), default_result_assigner(), default_rf(), mbo_defaults

---

loop_function  

**Loop Functions for Bayesian Optimization**

**Description**

Loop functions determine the behavior of the Bayesian Optimization algorithm on a global level. For an overview of readily available loop functions, see as.data.table(mlr_loop_functions).

In general, a loop function is simply a decorated member of the S3 class loop_function. Attributes must include: id (id of the loop function), label (brief description), instance ("single-crit" and or "multi_crit"), and man (link to the manual page).

As an example, see, e.g., bayesopt_ego.

See Also

Other Loop Function: mlr_loop_functions, mlr_loop_functions_ego, mlr_loop_functions_emo, mlr_loop_functions_mpcl, mlr_loop_functions_parego, mlr_loop_functions_smsego

---

mbo_defaults  

**Defaults for OptimizerMbo**

**Description**

The following defaults are set for OptimizerMbo during optimization if the respective fields are not set during initialization.

- Optimization Loop: default_loop_function

- Surrogate: default_surrogate

- Acquisition Function: default_acqfunction

- Acqfun Optimizer: default_acqoptimizer

- Result Assigner: default_result_assigner

See Also

Other mbo_defaults: default_acqfunction(), default_acqoptimizer(), default_gp(), default_loop_function(), default_result_assigner(), default_rf(), default_surrogate()
Dictionary of Acquisition Functions

Description
A simple mlr3misc::Dictionary storing objects of class AcqFunction. Each acquisition function has an associated help page, see mlr_acqfunctions_[id].
For a more convenient way to retrieve and construct an acquisition function, see acqf().

Format
R6::R6Class object inheriting from mlr3misc::Dictionary.

Methods
See mlr3misc::Dictionary.

See Also
Sugar function: acqf()
Other Dictionary: mlr_loop_functions, mlr_result_assigners
Other Acquisition Function: AcqFunction, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego

Examples
library(data.table)
as.data.table(mlr_acqfunctions)
acqf("ei")

Acquisition Function Augmented Expected Improvement

Description
Augmented Expected Improvement. Useful when working with noisy objectives. Currently only works correctly with "regr.km" as surrogate model and nugget.estim = TRUE or given.

Dictionary
This AcqFunction can be instantiated via the dictionary mlr_acqfunctions or with the associated sugar function acqf():

mlr_acqfunctions$get("aei")
acqf("aei")
Parameters

• "c" (numeric(1))
  Constant c as used in Formula (14) of Huang (2012) to reflect the degree of risk aversion. Defaults to 1.

Super classes

bbotk::Objective -> mlr3mbo::AcqFunction -> AcqFunctionAEI

Public fields

  y_effective_best (numeric(1))
  Best effective objective value observed so far. In the case of maximization, this already includes the necessary change of sign.

  noise_var (numeric(1))
  Estimate of the variance of the noise. This corresponds to the nugget estimate when using a mlr3learners as surrogate model.

Methods

Public methods:

• AcqFunctionAEI$new()
• AcqFunctionAEI$update()
• AcqFunctionAEI$clone()

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

Usage:
AcqFunctionAEI$new(surrogate = NULL, c = 1)

Arguments:
surrogate (NULL | SurrogateLearner).
c (numeric(1)).

Method update(): Updates acquisition function and sets y_effective_best and noise_var.

Usage:
AcqFunctionAEI$update()

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

Usage:
AcqFunctionAEI$clone(deep = FALSE)

Arguments:
deepl Whether to make a deep clone.

References

See Also

Other Acquisition Function: AcqFunction, mlr_acqfunctions, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvg, mlr_acqfunctions_ei, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego

Examples

```r
if (requireNamespace("mlr3learners") &
   requireNamespace("DiceKriging") &
   requireNamespace("rgenoud")) {
  library(bbotk)
  library(paradox)
  library(mlr3learners)
  library(data.table)

  set.seed(2906)
  fun = function(xs) {
    list(y = xs$x ^ 2 + rnorm(length(xs$x), mean = 0, sd = 1))
  }
  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun,
                               domain = domain,
                               codomain = codomain,
                               properties = "noisy")

  instance = OptimInstanceSingleCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))

  instance$eval_batch(data.table(x = c(-6, -5, 3, 9)))

  learner = lrn("regr.km",
                 covtype = "matern5_2",
                 optim.method = "gen",
                 nugget.estim = TRUE,
                 jitter = 1e-12,
                 control = list(trace = FALSE))

  surrogate = srlrn(learner, archive = instance$archive)

  acq_function = acqf("aei", surrogate = surrogate)

  acq_function$surrogate$update()
  acq_function$update()
  acq_function$eval_dt(data.table(x = c(-1, 0, 1)))
}
```
Acquisition Function Confidence Bound

Description

Lower / Upper Confidence Bound.

Dictionary

This AcqFunction can be instantiated via the dictionary mlr_acqfunctions or with the associated sugar function acqf():

```r
mlr_acqfunctions$get("cb")
acqf("cb")
```

Parameters

- "lambda" (numeric(1))
  
  λ value used for the confidence bound. Defaults to 2.

Super classes

bbotk::Objective -> mlr3mbo::AcqFunction -> AcqFunctionCB

Methods

Public methods:
- AcqFunctionCB$new()
- AcqFunctionCB$clone()

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

Usage:
```r
AcqFunctionCB$new(surrogate = NULL, lambda = 2)
```

Arguments:
- surrogate (NULL | SurrogateLearner).
- lambda (numeric(1)).

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

Usage:
```r
AcqFunctionCB$clone(deep = FALSE)
```

Arguments:
- deep Whether to make a deep clone.
References


See Also

Other Acquisition Function: AcqFunction, mlr_acqfunctions, mlr_acqfunctions_aei, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego

Examples

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {
  library(bbotk)
  library(paradox)
  library(mlr3learners)
  library(data.table)

  fun = function(xs) {
    list(y = xs$x ^ 2)
  }
  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceSingleCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))

  instance$eval_batch(data.table(x = c(-6, -5, 3, 9)))

  learner = default_gp()

  surrogate = srlrn(learner, archive = instance$archive)

  acq_function = acqf("cb", surrogate = surrogate, lambda = 3)

  acq_function$surrogate$update()
  acq_function$eval_dt(data.table(x = c(-1, 0, 1)))
}
```
**Description**

Exact Expected Hypervolume Improvement. Calculates the exact expected hypervolume improvement in the case of two objectives. In the case of optimizing more than two objective functions, AcqFunctionEHVIGH can be used. See Emmerich et al. (2016) for details.

**Super classes**

`bbotk::Objective` -> `mlr3mbo::AcqFunction` -> `AcqFunctionEHVI`

**Public fields**

- `ys_front (matrix())`
  Approximated Pareto front. Sorted by the first objective. Signs are corrected with respect to assuming minimization of objectives.

- `ref_point (numeric())`
  Reference point. Signs are corrected with respect to assuming minimization of objectives.

- `ys_front_augmented (matrix())`
  Augmented approximated Pareto front. Sorted by the first objective. Signs are corrected with respect to assuming minimization of objectives.

**Methods**

**Public methods:**

- `AcqFunctionEHVI$new()`
- `AcqFunctionEHVI$update()`
- `AcqFunctionEHVI$clone()`

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

*Usage:*

`AcqFunctionEHVI$new(surrogate = NULL)`

*Arguments:*

- `surrogate` (NULL | SurrogateLearnerCollection).

**Method** `update()`: Updates acquisition function and sets `ys_front`, `ref_point`.

*Usage:*

`AcqFunctionEHVI$update()`

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

*Usage:*

`AcqFunctionEHVI$clone(deep = FALSE)`

*Arguments:*

- `deep` Whether to make a deep clone.
References


See Also

Other Acquisition Function: AcqFunction, mlr_acqfunctions, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego

Examples

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {
  library(bbotk)
  library(paradox)
  library(mlr3learners)
  library(data.table)

  fun = function(xs) {
    list(y1 = xs$x^2, y2 = (xs$x - 2) ^ 2)
  }
  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y1 = p_dbl(tags = "minimize"), y2 = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceMultiCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))

  instance$eval_batch(data.table(x = c(-6, -5, 3, 9)))

  learner = default_gp()

  surrogate = srlrn(list(learner, learner$clone(deep = TRUE)), archive = instance$archive)

  acq_function = acqf("ehvi", surrogate = surrogate)

  acq_function$surrogate$update()
  acq_function$update()
  acq_function$eval_dt(data.table(x = c(-1, 0, 1)))
}
```

---

**mlr_acqfunctions_ehvigh**

*Acquisition Function Expected Hypervolume Improvement via Gauss-Hermite Quadrature*
Description

In the case of optimizing only two objective functions AcqFunctionEHVI is to be preferred.

Parameters

• "k" (integer(1))
  Number of nodes per objective used for the numerical integration via Gauss-Hermite quadrature. Defaults to 15. For example, if two objectives are to be optimized, the total number of nodes will therefore be 225 per default. Changing this value after construction requires a call to $update() to update the $gh_data field.

• "r" (numeric(1))
  Pruning rate between 0 and 1 that determines the fraction of nodes of the Gauss-Hermite quadrature rule that are ignored based on their weight value (the nodes with the lowest weights being ignored). Default is 0.2. Changing this value after construction does not require a call to $update().

Super classes

bbotk::Objective -> mlr3mbo::AcqFunction -> AcqFunctionEHVIGH

Public fields

ys_front (matrix())
  Approximated Pareto front. Signs are corrected with respect to assuming minimization of objectives.

ref_point (numeric())
  Reference point. Signs are corrected with respect to assuming minimization of objectives.

hypervolume (numeric(1)). Current hypervolume of the approximated Pareto front with respect to the reference point.

gh_data (matrix())
  Data required for the Gauss-Hermite quadrature rule in the form of a matrix of dimension (k x 2). Each row corresponds to one Gauss-Hermite node (column "x") and corresponding weight (column "w"). Computed via fastGHQuad::gaussHermiteData. Nodes are scaled by a factor of sqrt(2) and weights are normalized under a sum to one constraint.

Methods

Public methods:

• AcqFunctionEHVIGH$new()
• AcqFunctionEHVIGH$update()
• AcqFunctionEHVIGH$clone()

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

Usage:
AcqFunctionEHVIGH$new(surrogate = NULL, k = 15L, r = 0.2)
Arguments:
surrogate (NULL | SurrogateLearnerCollection).
k (integer(1)).
r (numeric(1)).

Method `update()`: Updates acquisition function and sets `ys_front`, `ref_point`, `hypervolume`, `gh_data`.

Usage:
AcqFunctionEHVIGH$update()

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

Usage:
AcqFunctionEHVIGH$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

References

See Also
Other Acquisition Function: AcqFunction, mlr_acqfunctions, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ei, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego

Examples
```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {
  library(bbotk)
  library(paradox)
  library(mlr3learners)
  library(data.table)

  fun = function(xs) {
    list(y1 = xs$x^2, y2 = (xs$x - 2) ^ 2)
  }
  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y1 = p_dbl(tags = "minimize"), y2 = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceMultiCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))
```
instance$eval_batch(data.table(x = c(-6, -5, 3, 9)))

learner = default_gp()

surrogate = srlrn(list(learner, learner$clone(deep = TRUE)), archive = instance$archive)

acq_function = acqf("ehvigh", surrogate = surrogate)

acq_function$surrogate$update()
acq_function$update()
acq_function$eval_dt(data.table(x = c(-1, 0, 1)))

mlr_acqfunctions_ei

Acquisition Function Expected Improvement

Description

Expected Improvement.

Dictionary

This AcqFunction can be instantiated via the dictionary mlr_acqfunctions or with the associated sugar function acqf():

mlr_acqfunctions$get("ei")
acqf("ei")

Super classes

bbotk::Objective -> mlr3mbo::AcqFunction -> AcqFunctionEI

Public fields

y_best (numeric(1))

Best objective function value observed so far. In the case of maximization, this already includes the necessary change of sign.

Methods

Public methods:

- AcqFunctionEI$new()
- AcqFunctionEI$update()
- AcqFunctionEI$clone()

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

Usage:
AcqFunctionEI$new(surrogate = NULL)

**Arguments:**
surrogate (NULL | SurrogateLearner).

**Method** update(): Updates acquisition function and sets y_best.

**Usage:**
AcqFunctionEI$update()

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

**Usage:**
AcqFunctionEI$clone(deep = FALSE)

**Arguments:**
depth Whether to make a deep clone.

**References**


**See Also**

Other Acquisition Function: AcqFunction, mlr_acqfunctions, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego

**Examples**

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {
  library(bbotk)
  library(paradox)
  library(mlr3learners)
  library(data.table)

  fun = function(xs) {
    list(y = xs$x ^ 2)
  }
  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceSingleCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))

  instance$eval_batch(data.table(x = c(-6, -5, 3, 9)))
  learner = default_gp()
```
surrogate = srlrn(learner, archive = instance$archive)

acq_function = acqf("ei", surrogate = surrogate)

acq_function$surrogate$update()
acq_function$update()
acq_function$eval_dt(data.table(x = c(-1, 0, 1)))

---

**mlr_acqfunctions_eips**  
*Acquisition Function Expected Improvement Per Second*

**Description**

Expected Improvement per Second.  

It is assumed that calculations are performed on an bbotk::OptimInstanceSingleCrit. Additionally to target values of the codomain that should be minimized or maximized, the bbotk::Objective of the bbotk::OptimInstanceSingleCrit should return time values. The column names of the target variable and time variable must be passed as cols_y in the order (target, time) when constructing the SurrogateLearnerCollection that is being used as a surrogate.

**Dictionary**

This AcqFunction can be instantiated via the dictionary mlr_acqfunctions or with the associated sugar function acqf():

```r
mlr_acqfunctions$get("eips")
acqf("eips")
```

**Super classes**

bbotk::Objective -> mlr3mbo::AcqFunction -> AcqFunctionEIPS

**Public fields**

`y_best` (numeric(1))

Best objective function value observed so far. In the case of maximization, this already includes the necessary change of sign.

**Active bindings**

`col_y` (character(1)).  
`col_time` (character(1)).
Methods

Public methods:

- AcqFunctionEIPS$new()
- AcqFunctionEIPS$update()
- AcqFunctionEIPS$clone()

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

Usage:
AcqFunctionEIPS$new(surrogate = NULL)

Arguments:
surrogate (NULL | SurrogateLearnerCollection).

**Method update():** Updates acquisition function and sets y_best.

Usage:
AcqFunctionEIPS$update()

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

Usage:
AcqFunctionEIPS$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

References


See Also

Other Acquisition Function: AcqFunction, mlr_acqfunctions, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_mean, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego

Examples

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {
  library(bboptk)
  library(paradox)
  library(mlr3learners)
  library(data.table)

  fun = function(xs) {
    list(y = xs$x ^ 2, time = abs(xs$x))
  }
```
domain = ps(x = p_dbl(lower = -10, upper = 10))
codomain = ps(y = p_dbl(tags = "minimize"), time = p_dbl(tags = "time"))
objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

instance = OptimInstanceSingleCrit$new(
  objective = objective,
  terminator = trm("evals", n_evals = 5))

instance$eval_batch(data.table(x = c(-6, -5, 3, 9)))

learner = default_gp()
surrogate = srlrn(list(learner, learner$clone(deep = TRUE)), archive = instance$archive)
surrogate$cols_y = c("y", "time")
acq_function = acqf("eips", surrogate = surrogate)
acq_function$surrogate$update()
acq_function$update()
acq_function$eval_dt(data.table(x = c(-1, 0, 1)))
}

mlr_acqfunctions_mean  Acquisition Function Mean

Description

Posterior Mean.

Dictionary

This AcqFunction can be instantiated via the dictionary mlr_acqfunctions or with the associated sugar function acqf():

mlr_acqfunctions$get("mean")
acqf("mean")

Super classes

bbotk::Objective -> mlr3mbo::AcqFunction -> AcqFunctionMean

Methods

Public methods:

- AcqFunctionMean$new()
- AcqFunctionMean$clone()

Method new(): Creates a new instance of this R6 class.
Usage:
AcqFunctionMean$new(surrogate = NULL)

Arguments:
surrogate (NULL | SurrogateLearner).

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

Usage:
AcqFunctionMean$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

See Also
Other Acquisition Function: AcqFunction, mlr_acqfunctions, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_eips, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego

Examples
if (requireNamespace("mlr3learners") & requireNamespace("DiceKriging") & requireNamespace("rgenoud")) {
  library(bbotk)
  library(paradox)
  library(mlr3learners)
  library(data.table)

  fun = function(xs) {
    list(y = xs$x ^ 2)
  }

  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceSingleCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))

  instance$eval_batch(data.table(x = c(-6, -5, 3, 9)))

  learner = default_gp()

  surrogate = srlrn(learner, archive = instance$archive)

  acq_function = acqf("mean", surrogate = surrogate)

  acq_function$surrogate$update()
  acq_function$update()
  acq_function$eval_dt(data.table(x = c(-1, 0, 1)))
}
mlr_acqfunctions_pi  Acquisition Function Probability of Improvement

Description

Probability of Improvement.

Dictionary

This AcqFunction can be instantiated via the dictionary mlr_acqfunctions or with the associated sugar function acqf():

```r
mlr_acqfunctions$get("pi")
acqf("pi")
```

Super classes

`bbotk::Objective` -> `mlr3mbo::AcqFunction` -> `AcqFunctionPI`

Public fields

`y_best` (numeric(1))

Best objective function value observed so far. In the case of maximization, this already includes the necessary change of sign.

Methods

Public methods:

- `AcqFunctionPI$new()`
- `AcqFunctionPI$update()`
- `AcqFunctionPI$clone()`

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

Usage:

```r
AcqFunctionPI$new(surrogate = NULL)
```

Arguments:

`surrogate` (NULL | SurrogateLearner).

Method `update()`: Updates acquisition function and sets `y_best`.

Usage:

```r
AcqFunctionPI$update()
```

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

Usage:

```r
AcqFunctionPI$clone(deep = FALSE)
```

Arguments:

deep  Whether to make a deep clone.
References


See Also

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions`, `mlr_acqfunctions_aei`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_ehvi`, `mlr_acqfunctions_ehvhig`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_sd`, `mlr_acqfunctions_smsego`

Examples

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {
  library(bbotk)
  library(paradox)
  library(mlr3learners)
  library(data.table)

  fun = function(xs) {
    list(y = xs$x ^ 2)
  }
  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceSingleCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))

  instance$eval_batch(data.table(x = c(-6, -5, 3, 9)))

  learner = default_gp()

  surrogate = srlrn(learner, archive = instance$archive)

  acq_function = acqf("pi", surrogate = surrogate)

  acq_function$surrogate$update()
  acq_function$update()
  acq_function$eval_dt(data.table(x = c(-1, 0, 1)))
}
```

---

**mlr_acqfunctions_sd**  
*Acquisition Function Standard Deviation*

**Description**

Posterior Standard Deviation.
Dictionary

This AcqFunction can be instantiated via the dictionary mlr_acqfunctions or with the associated sugar function acqf():

```r
mlr_acqfunctions$"get("sd")
acqf("sd")
```

Super classes

`bbotk::Objective` -> `mlr3mbo::AcqFunction` -> `AcqFunctionSD`

Methods

Public methods:

- `AcqFunctionSD$new()`
- `AcqFunctionSD$clone()`

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

Usage:

```r
AcqFunctionSD$new(surrogate = NULL)
```

Arguments:

`surrogate` (NULL | SurrogateLearner).

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

Usage:

```r
AcqFunctionSD$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions`, `mlr_acqfunctions_aei`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_ehvi`, `mlr_acqfunctions_ehvigh`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_smsego`

Examples

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {
  library(bbotk)
  library(paradox)
  library(mlr3learners)
  library(data.table)

  fun = function(xs) {
    list(y = xs$x ^ 2)
  }
```
Acquisition Function SMS-EGO

Description

S-Metric Selection Evolutionary Multi-Objective Optimization Algorithm Acquisition Function.

Parameters

• "lambda" (numeric(1))
  \( \lambda \) value used for the confidence bound. Defaults to 1. Based on confidence = \((1 - 2 * dnorm(lambda))^m\) you can calculate a lambda for a given confidence level, see Ponweiser et al. (2008).

• "epsilon" (numeric(1))
  \( \epsilon \) used for the additive epsilon dominance. Can either be a single numeric value > 0 or NULL (default). In the case of being NULL, an epsilon vector is maintained dynamically as described in Horn et al. (2015).

Super classes

bbotk::Objective -> mlr3mbo::AcqFunction -> AcqFunctionSmsEgo
Public fields

ys_front (matrix())
Approximated Pareto front. Signs are corrected with respect to assuming minimization of objectives.

ref_point (numeric())
Reference point. Signs are corrected with respect to assuming minimization of objectives.

epsilon (numeric())
Epsilon used for the additive epsilon dominance.

progress (numeric())
Optimization progress (typically, the number of function evaluations left). Note that this requires the bbotk::OptimInstance to be terminated via a bbotk::TerminatorEvals.

Methods

Public methods:

• AcqFunctionSmsEgo$new()
• AcqFunctionSmsEgo$update()
• AcqFunctionSmsEgo$clone()

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

Usage:
AcqFunctionSmsEgo$new(surrogate = NULL, lambda = 1, epsilon = NULL)

Arguments:
surrogate (NULL | SurrogateLearnerCollection).
lambda (numeric(1)).
epsilon (NULL | numeric(1)).

Method update(): Updates acquisition function and sets ys_front, ref_point, epsilon.

Usage:
AcqFunctionSmsEgo$update()

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

Usage:
AcqFunctionSmsEgo$clone(deep = FALSE)

Arguments:
depth Whether to make a deep clone.

References


mlr_loop_functions

Dictionary of Loop Functions

Description

A simple mlr3misc::Dictionary storing objects of class loop_function. Each loop function has an associated help page, see mlr_loop_functions_[id].

Retrieves object with key key from the dictionary. Additional arguments must be named and are passed to the constructor of the stored object.

See Also

Other Acquisition Function: AcqFunction, mlr_acqfunctions, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_pi, mlr_acqfunctions_sd

Examples

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {
library(bbotk)
library(paradox)
library(mlr3learners)
library(data.table)

fun = function(xs) {
    list(y1 = xs$x^2, y2 = (xs$x - 2) ^ 2)
}
domain = ps(x = p_dbl(lower = -10, upper = 10))
codomain = ps(y1 = p_dbl(tags = "minimize"), y2 = p_dbl(tags = "minimize"))
objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

instance = OptimInstanceMultiCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))

instance$eval_batch(data.table(x = c(-6, -5, 3, 9)))

learner = default_gp()

surrogate = srlrn(list(learner, learner$clone(deep = TRUE)), archive = instance$archive)

acq_function = acqf("smsego", surrogate = surrogate)

acq_function$surrogate$update()
acq_function$progress = 5 - 4 # n_evals = 5 and 4 points already evaluated
acq_function$update()

acq_function$eval_dt(data.table(x = c(-1, 0, 1)))
}
```
mlr_loop_functions_ego

Arguments

key (character(1)).

Passed down to constructor.

Format

R6::R6Class object inheriting from mlr3misc::Dictionary.

Value

Object with corresponding key.

Methods

See mlr3misc::Dictionary.

See Also

Other Dictionary: mlr_acqfunctions, mlr_result_assigners

Other Loop Function: loop_function, mlr_loop_functions_ego, mlr_loop_functions_emo, mlr_loop_functions_mpcl, mlr_loop_functions_parego, mlr_loop_functions_smseg

Examples

library(data.table)
as.data.table(mlr_loop_functions)

mlr_loop_functions_ego

Sequential Single-Objective Bayesian Optimization

Description

Loop function for sequential single-objective Bayesian Optimization. Normally used inside an OptimizerMbo.

In each iteration after the initial design, the surrogate and acquisition function are updated and the next candidate is chosen based on optimizing the acquisition function.

Usage

bayesopt_ego(
  instance,
  surrogate,
  acq_function,
  acq_optimizer,
  init_design_size = NULL,
  random_interleave_iter = 0L
)

Arguments

instance (bbotk::OptimInstanceSingleCrit)
The bbotk::OptimInstanceSingleCrit to be optimized.

surrogate (Surrogate)
Surrogate to be used as a surrogate. Typically a SurrogateLearner.

acq_function (AcqFunction)
AcqFunction to be used as acquisition function.

acq_optimizer (AcqOptimizer)
AcqOptimizer to be used as acquisition function optimizer.

init_design_size (NULL | integer(1))
Size of the initial design. If NULL and the bbotk::Archive contains no evaluations,
4 * d is used with d being the dimensionality of the search space. Points are
generated via a Sobol sequence.

random_interleave_iter (integer(1))
Every random_interleave_iter iteration (starting after the initial design), a
point is sampled uniformly at random and evaluated (instead of a model based
proposal). For example, if random_interleave_iter = 2, random interleaving
is performed in the second, fourth, sixth, ... iteration. Default is 0, i.e., no
random interleaving is performed at all.

Value

invisible(instance)
The original instance is modified in-place and returned invisible.

Note

• The acq_function$surrogate, even if already populated, will always be overwritten by the
surrogate.
• The acq_optimizer$acq_function, even if already populated, will always be overwritten
by acq_function.
• The surrogate$archive, even if already populated, will always be overwritten by the bbotk::Archive
of the bbotk::OptimInstanceSingleCrit.

References

Expensive Black-Box Functions.” Journal of Global optimization, 13(4), 455–492.
• Snoek, Jasper, Larochelle, Hugo, Adams, P R (2012). “Practical Bayesian Optimization of

See Also

Other Loop Function: loop_function, mlr_loop_functions, mlr_loop_functions_emo, mlr_loop_functions_mpcl,
mlr_loop_functions_parego, mlr_loop_functions_smsego
Examples

```
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {

library(bbotk)
library(paradox)
library(mlr3learners)

fun = function(xs) {
  list(y = xs$x ^ 2)
}
domain = ps(x = p_dbl(lower = -10, upper = 10))
codomain = ps(y = p_dbl(tags = "minimize"))
objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

instance = OptimInstanceSingleCrit$new(
  objective = objective,
  terminator = trm("evals", n_evals = 5))
surrogate = default_surrogate(instance)

acq_function = acqf("ei")
acq_optimizer = acqo(
  optimizer = opt("random_search", batch_size = 100),
  terminator = trm("evals", n_evals = 100))

optimizer = opt("mbo",
  loop_function = bayesopt_ego,
  surrogate = surrogate,
  acq_function = acq_function,
  acq_optimizer = acq_optimizer)

optimizer$optimize(instance)

# expected improvement per second example
fun = function(xs) {
  list(y = xs$x ^ 2, time = abs(xs$x))
}
domain = ps(x = p_dbl(lower = -10, upper = 10))
codomain = ps(y = p_dbl(tags = "minimize"), time = p_dbl(tags = "time"))
objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

instance = OptimInstanceSingleCrit$new(
  objective = objective,
  terminator = trm("evals", n_evals = 5))
surrogate = default_surrogate(instance, n_learner = 2)
surrogate$cols_y = c("y", "time")
```
mlr_loop_functions_emo

Sequential Multi-Objective Bayesian Optimization

Description

Loop function for sequential multi-objective Bayesian Optimization. Normally used inside an OptimizerMbo. The conceptual counterpart to mlr_loop_functions_ego.

In each iteration after the initial design, the surrogate and acquisition function are updated and the next candidate is chosen based on optimizing the acquisition function.

Usage

bayesopt_emo(
    instance,
    surrogate,
    acq_function,
    acq_optimizer,
    init_design_size = NULL,
    random_interleave_iter = 0L
)

Arguments

instance  (bbotk::OptimInstanceMultiCrit)
The bbotk::OptimInstanceMultiCrit to be optimized.
surrogate  (SurrogateLearnerCollection)
SurrogateLearnerCollection to be used as a surrogate.
acq_function  (AcqFunction)
AcqFunction to be used as acquisition function.
acq_optimizer  (AcqOptimizer)
AcqOptimizer to be used as acquisition function optimizer.
init_design_size  (NULL | integer(1))
Size of the initial design. If NULL and the bbotk::Archive contains no evaluations, $4 \times d$ is used with $d$ being the dimensionality of the search space. Points are generated via a Sobol sequence.
random_interleave_iter

(integer(1))

Every random_interleave_iter iteration (starting after the initial design), a point is sampled uniformly at random and evaluated (instead of a model based proposal). For example, if random_interleave_iter = 2, random interleaving is performed in the second, fourth, sixth, ... iteration. Default is 0, i.e., no random interleaving is performed at all.

Value

invisible(instance)

The original instance is modified in-place and returned invisible.

Note

- The acq_function$surrogate, even if already populated, will always be overwritten by the surrogate.
- The acq_optimizer$acq_function, even if already populated, will always be overwritten by acq_function.
- The surrogate$archive, even if already populated, will always be overwritten by the bbotk::Archive of the bbotk::OptimInstanceMultiCrit.

See Also

Other Loop Function: loop_function, mlr_loop_functions, mlr_loop_functions_ego, mlr_loop_functions_mpcl, mlr_loop_functions_parego, mlr_loop_functions_smsego

Examples

if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {

library(bbotk)
library(paradox)
library(mlr3learners)

fun = function(xs) {
  list(y1 = xs$x^2, y2 = (xs$x - 2)^2)
}
domain = ps(x = p_dbl(lower = -10, upper = 10))
codomain = ps(y1 = p_dbl(tags = "minimize"), y2 = p_dbl(tags = "minimize"))
objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

instance = OptimInstanceMultiCrit$new(
  objective = objective,
  terminator = trm("evals", n_evals = 5))

surrogate = default_surrogate(instance)
acq_function = acqf("ehvi")

acq_optimizer = acqo(
    optimizer = opt("random_search", batch_size = 100),
    terminator = trm("evals", n_evals = 100))

optimizer = opt("mbo",
    loop_function = bayesopt_emo,
    surrogate = surrogate,
    acq_function = acq_function,
    acq_optimizer = acq_optimizer)

optimizer$optimize(instance)
}

---

Single-Objective Bayesian Optimization via Multipoint Constant Liar

**Description**

Loop function for single-objective Bayesian Optimization via multipoint constant liar. Normally used inside an OptimizerMbo.

In each iteration after the initial design, the surrogate and acquisition function are updated. The acquisition function is then optimized, to find a candidate but instead of evaluating this candidate, the objective function value is obtained by applying the liar function to all previously obtained objective function values. This is repeated \( q - 1 \) times to obtain a total of \( q \) candidates that are then evaluated in a single batch.

**Usage**

```r
bayesopt_mpcl(
    instance,
    surrogate,
    acq_function,
    acq_optimizer,
    init_design_size = NULL,
    q = 2L,
    liar = mean,
    random_interleave_iter = 0L
)
```

**Arguments**

- `instance` *(bbotk::OptimInstanceSingleCrit)*
  The bbotk::OptimInstanceSingleCrit to be optimized.
surrogate (Surrogate)
Surrogate to be used as a surrogate. Typically a SurrogateLearner.

acq_function (AcqFunction)
AcqFunction to be used as acquisition function.

acq_optimizer (AcqOptimizer)
AcqOptimizer to be used as acquisition function optimizer.

init_design_size

(NULL | integer(1))
Size of the initial design. If NULL and the bbotk::Archive contains no evaluations, 4 * d is used with d being the dimensionality of the search space. Points are generated via a Sobol sequence.

q (integer(1))
Batch size > 1. Default is 2.

liar (function)
Any function accepting a numeric vector as input and returning a single numeric output. Default is mean. Other sensible functions include min (or max, depending on the optimization direction).

random_interleave_iter

(integer(1))
Every random_interleave_iter iteration (starting after the initial design), a point is sampled uniformly at random and evaluated (instead of a model based proposal). For example, if random_interleave_iter = 2, random interleaving is performed in the second, fourth, sixth, ... iteration. Default is 0, i.e., no random interleaving is performed at all.

Value

invisible(instance)
The original instance is modified in-place and returned invisible.

Note

• The acq_function$surrogate, even if already populated, will always be overwritten by the surrogate.
• The acq_optimizer$acq_function, even if already populated, will always be overwritten by acq_function.
• The surrogate$archive, even if already populated, will always be overwritten by the bbotk::Archive of the bbotk::OptimInstanceSingleCrit.
• To make use of parallel evaluations in the case of ‘q > 1, the objective function of the bbotk::OptimInstanceSingleCrit must be implemented accordingly.

References

See Also

Other Loop Function: `loop_function`, `mlr_loop_functions`, `mlr_loop_functions_ego`, `mlr_loop_functions_emo`, `mlr_loop_functions_parego`, `mlr_loop_functions_smsego`

Examples

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {

  library(bbotk)
  library(paradox)
  library(mlr3learners)

  fun = function(xs) {
    list(y = xs$x^2)
  }
  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceSingleCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 7))

  surrogate = default_surrogate(instance)

  acq_function = acqf("ei")
  acq_optimiser = acqo(
    optimizer = opt("random_search", batch_size = 100),
    terminator = trm("evals", n_evals = 100))

  optimizer = opt("mbo",
    loop_function = bayesopt_mpcl,
    surrogate = surrogate,
    acq_function = acq_function,
    acq_optimiser = acq_optimiser,
    args = list(q = 3))

  optimizer$optimize(instance)
}
```
Description

Loop function for multi-objective Bayesian Optimization via ParEGO. Normally used inside an OptimizerMbo.

In each iteration after the initial design, the observed objective function values are normalized and q candidates are obtained by scalarizing these values via the augmented Tchebycheff function, updating the surrogate with respect to these scalarized values and optimizing the acquisition function.

Usage

bayesopt_parego(
  instance, surrogate, acq_function, acq_optimizer, init_design_size = NULL, q = 1L, s = 100L, rho = 0.05, random_interleave_iter = 0L
)

Arguments

instance (bbotk::OptimInstanceMultiCrit)
The bbotk::OptimInstanceMultiCrit to be optimized.
surrogate (SurrogateLearner)
SurrogateLearner to be used as a surrogate.
acq_function (AcqFunction)
AcqFunction to be used as acquisition function.
acq_optimizer (AcqOptimizer)
AcqOptimizer to be used as acquisition function optimizer.
init_design_size (NULL | integer(1))
Size of the initial design. If NULL and the bbotk::Archive contains no evaluations, 4 * d is used with d being the dimensionality of the search space. Points are generated via a Sobol sequence.
q (integer(1))
Batch size, i.e., the number of candidates to be obtained for a single batch. Default is 1.
s (integer(1))
s in Equation 1 in Knowles (2006). Determines the total number of possible random weight vectors. Default is 100.
rho (numeric(1))
ρ in Equation 2 in Knowles (2006) scaling the linear part of the augmented Tchebycheff function. Default is 0.05.
random_interleave_iter

(integer(1))

Every random_interleave_iter iteration (starting after the initial design), a point is sampled uniformly at random and evaluated (instead of a model based proposal). For example, if random_interleave_iter = 2, random interleaving is performed in the second, fourth, sixth, ... iteration. Default is 0, i.e., no random interleaving is performed at all.

Value

invisible(instance)

The original instance is modified in-place and returned invisible.

Note

- The acq_function$surrogate, even if already populated, will always be overwritten by the surrogate.
- The acq_optimizer$acq_function, even if already populated, will always be overwritten by acq_function.
- The surrogate$archive, even if already populated, will always be overwritten by the bbotk::Archive of the bbotk::OptimInstanceMultiCrit.
- The scalarizations of the objective function values are stored as the y_scal column in the bbotk::Archive of the bbotk::OptimInstanceMultiCrit.
- To make use of parallel evaluations in the case of `q > 1, the objective function of the bbotk::OptimInstanceMultiCrit must be implemented accordingly.

References


See Also

Other Loop Function: loop_function, mlr_loop_functions, mlr_loop_functions_ego, mlr_loop_functions_emo, mlr_loop_functions_mpc1, mlr_loop_functions_smsego

Examples

```r
if (requireNamespace("mlr3learners") & requireNamespace("DiceKriging") & requireNamespace("rgenoud")) {
  library(bbotk)
  library(paradox)
  library(mlr3learners)

  fun = function(xs) {
    # Code
  }
}
```
mlr_loop_functions_smsego

Sequential Multi-Objective Bayesian Optimization via SMS-EGO

Description

Loop function for sequential multi-objective Bayesian Optimization via SMS-EGO. Normally used inside an OptimizerMbo.

In each iteration after the initial design, the surrogate and acquisition function (mlr_acqfunctions_smsego) are updated and the next candidate is chosen based on optimizing the acquisition function.

Usage

bayesopt_smsego(
  instance,
  surrogate,
  acq_function,
  acq_optimizer,
  init_design_size = NULL,
  random_interleave_iter = 0L
)
Arguments

instance (bbotk::OptimInstanceMultiCrit)
The bbotk::OptimInstanceMultiCrit to be optimized.
surrogate (SurrogateLearnerCollection)
SurrogateLearnerCollection to be used as a surrogate.
acq_function (mlr_acqfunctions_smsego)
mlr_acqfunctions_smsego to be used as acquisition function.
acq_optimizer (AcqOptimizer)
AcqOptimizer to be used as acquisition function optimizer.
init_design_size (NULL | integer(1))
Size of the initial design. If NULL and the bbotk::Archive contains no evaluations, 4 * d is used with d being the dimensionality of the search space. Points are generated via a Sobol sequence.
random_interleave_iter (integer(1))
Every random_interleave_iter iteration (starting after the initial design), a point is sampled uniformly at random and evaluated (instead of a model based proposal). For example, if random_interleave_iter = 2, random interleaving is performed in the second, fourth, sixth, ... iteration. Default is 0, i.e., no random interleaving is performed at all.

Value

invisible(instance)
The original instance is modified in-place and returned invisible.

Note

• The acq_function$surrogate, even if already populated, will always be overwritten by the surrogate.
• The acq_optimizer$acq_function, even if already populated, will always be overwritten by acq_function.
• The surrogate$archive, even if already populated, will always be overwritten by the bbotk::Archive of the bbotk::OptimInstanceMultiCrit.
• Due to the iterative computation of the epsilon within the mlr_acqfunctions_smsego, requires the bbotk::Terminator of the bbotk::OptimInstanceMultiCrit to be a bbotk::TerminatorEvals.

References

See Also

Other Loop Function: `loop_function`, `mlr_loop_functions`, `mlr_loop_functions_ego`, `mlr_loop_functions_emo`, `mlr_loop_functions_mpcl`, `mlr_loop_functions_parego`

Examples

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {

  library(bbotk)
  library(paradox)
  library(mlr3learners)

  fun = function(xs) {
    list(y1 = xs$x^2, y2 = (xs$x - 2)^2)
  }
  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y1 = p_dbl(tags = "minimize"), y2 = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceMultiCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))

  surrogate = default_surrogate(instance)

  acq_function = acqf("smsego")
  acq_optimizer = acqo(
    optimizer = opt("random_search", batch_size = 100),
    terminator = trm("evals", n_evals = 100))

  optimizer = opt("mbo",
    loop_function = bayesopt_smsego,
    surrogate = surrogate,
    acq_function = acq_function,
    acq_optimizer = acq_optimizer)

  optimizer$optimize(instance)
}
```
OptimizerMbo class that implements Model Based Optimization (MBO). The implementation follows a modular layout relying on a loop_function determining the MBO flavor to be used, e.g., bayesopt_ego for sequential single-objective Bayesian Optimization, a Surrogate, an AcqFunction, e.g., mlr_acqfunctions_ei for Expected Improvement and an AcqOptimizer.

MBO algorithms are iterative optimization algorithms that make use of a continuously updated surrogate model built for the objective function. By optimizing a comparably cheap to evaluate acquisition function defined on the surrogate prediction, the next candidate is chosen for evaluation. Detailed descriptions of different MBO flavors are provided in the documentation of the respective loop_function.

Termination is handled via a bbotk::Terminator part of the bbotk::OptimInstance to be optimized. Note that in general the Surrogate is updated one final time on all available data after the optimization process has terminated. However, in certain scenarios this is not always possible or meaningful, e.g., when using bayesopt_parego() for multi-objective optimization which uses a surrogate that relies on a scalarization of the objectives. It is therefore recommended to manually inspect the Surrogate after optimization if it is to be used, e.g., for visualization purposes to make sure that it has been properly updated on all available data. If this final update of the Surrogate could not be performed successfully, a warning will be logged.

Archive

The bbotk::Archive holds the following additional columns that are specific to MBO algorithms:

- [acq_function$id](numeric(1))
  The value of the acquisition function.
- .already_evaluated(logical(1))
  Whether this point was already evaluated. Depends on the skip_already_evaluated parameter of the AcqOptimizer.

Super class

bbotk::Optimizer -> OptimizerMbo

Active bindings

- loop_function (loop_function | NULL)
  Loop function determining the MBO flavor.
- surrogate (Surrogate | NULL)
  The surrogate.
- acq_function (AcqFunction | NULL)
  The acquisition function.
- acq_optimizer (AcqOptimizer | NULL)
  The acquisition function optimizer.
- args (named list())
  Further arguments passed to the loop_function. For example, random_interleave_iter.
result_assigner (ResultAssigner | NULL)
The result assigner.

param_classes (character())
Supported parameter classes that the optimizer can optimize. Determined based on the surrogate and the acq_optimizer. This corresponds to the values given by a paradox::ParamSet's $class field.

properties (character())
Set of properties of the optimizer. Must be a subset of bbotk_reflections$optimizer_properties. MBO in principle is very flexible and by default we assume that the optimizer has all properties. When fully initialized, properties are determined based on the loop_function and surrogate.

packages (character())
Set of required packages. A warning is signaled prior to optimization if at least one of the packages is not installed, but loaded (not attached) later on-demand via requireNamespace(). Required packages are determined based on the acq_function, surrogate and the acq_optimizer.

Methods

Public methods:

- `OptimizerMbo$new()`
- `OptimizerMbo$print()`
- `OptimizerMbo$reset()`
- `OptimizerMbo$clone()`

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

If surrogate is NULL and the acq_function$surrogate field is populated, this Surrogate is used. Otherwise, default_surrogate(instance) is used. If acq_function is NULL and the acq_optimizer$acq_function field is populated, this AcqFunction is used (and therefore its $surrogate if populated; see above). Otherwise default_acqfunction(instance) is used. If acq_optimizer is NULL, default_acqoptimizer(instance) is used.

Even if already initialized, the surrogate$archive field will always be overwritten by the bbotk::Archive of the current bbotk::OptimInstance to be optimized.

For more information on default values for loop_function, surrogate, acq_function and acqOptimizer, see ?mbo_defaults.

Usage:

```
OptimizerMbo$new(
  loop_function = NULL,
  surrogate = NULL,
  acq_function = NULL,
  acq_optimizer = NULL,
  args = NULL,
  result_assigner = NULL
)
```

Arguments:

- `loop_function (loop_function | NULL)`
  Loop function determining the MBO flavor.
surrogate (Surrogate | NULL)
  The surrogate.
acq_function (AcqFunction | NULL)
  The acquisition function.
acq_optimizer (AcqOptimizer | NULL)
  The acquisition function optimizer.
args (named list())
  Further arguments passed to the loop_function. For example, random_interleave_iter.
result_assigner (ResultAssigner | NULL)
  The result assigner.

**Method** print(): Print method.

*Usage:*
OptimizerMbo$print()

*Returns:* (character()).

**Method** reset(): Reset the optimizer. Sets the following fields to NULL: loop_function, surrogate, acq_function, acq_optimizer, args, result_assigner

*Usage:*
OptimizerMbo$reset()

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

*Usage:*
OptimizerMbo$clone(deep = FALSE)

*Arguments:*
depth Whether to make a deep clone.

**Examples**

```r
if (requireNamespace("mlr3learners") & requireNamespace("DiceKriging") & requireNamespace("rgenoud")) {

  library(bbotk)
  library(paradox)
  library(mlr3learners)

  # single-objective EGO
  fun = function(xs) {
    list(y = xs$x ^ 2)
  }
  domain = ps(x = p_dbl(lower = -10, upper = 10))
  codomain = ps(y = p_dbl(tags = "minimize"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceSingleCrit$new(
    objective = objective,
```
terminator = trm("evals", n_evals = 5))

surrogate = default_surrogate(instance)

acq_function = acqf("ei")

acq_optimizer = acqo(
  optimizer = opt("random_search", batch_size = 100),
  terminator = trm("evals", n_evals = 100))

optimizer = opt("mbo",
  loop_function = bayesopt_ego,
  surrogate = surrogate,
  acq_function = acq_function,
  acq_optimizer = acq_optimizer)

optimizer$optimize(instance)

# multi-objective ParEGO
fun = function(xs) {
  list(y1 = xs$x^2, y2 = (xs$x - 2) ^ 2)
}

domain = ps(x = p_dbl(lower = -10, upper = 10))
codomain = ps(y1 = p_dbl(tags = "minimize"), y2 = p_dbl(tags = "minimize"))
objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

instance = OptimInstanceMultiCrit$new(
  objective = objective,
  terminator = trm("evals", n_evals = 5))

optimizer = opt("mbo",
  loop_function = bayesopt_parego,
  surrogate = surrogate,
  acq_function = acq_function,
  acq_optimizer = acq_optimizer)

optimizer$optimize(instance)

---

**mle_result_assigners**  
*Dictionary of Result Assigners*

**Description**

A simple `mlr3misc::Dictionary` storing objects of class `ResultAssigner`. Each acquisition function has an associated help page, see `mle_result_assigners_[id]`.

For a more convenient way to retrieve and construct an acquisition function, see `ras()`.
Format

\texttt{R6::R6Class} object inheriting from \texttt{mlr3misc::Dictionary}.

Methods

See \texttt{mlr3misc::Dictionary}.

See Also

Sugar function: \texttt{ras()}

Other Dictionary: \texttt{mlr_acqfunctions, mlr_loop_functions}

Other Result Assigner: \texttt{ResultAssigner, mlr_result_assigners_archive, mlr_result_assigners_surrogate}

Examples

\begin{verbatim}
library(data.table)
as.data.table(mlr_result_assigners)
ras("archive")
\end{verbatim}

---

\textbf{mlr_result_assigners_archive}

\textit{Result Assigner Based on the Archive}

Description

Result assigner that chooses the final point(s) based on all evaluations in the \texttt{bbotk::Archive}. This mimics the default behavior of any \texttt{bbotk::Optimizer}.

Super class

\texttt{mlr3mbo::ResultAssigner} \rightarrow \texttt{ResultAssignerArchive}

Active bindings

packages (character())

Set of required packages. A warning is signaled if at least one of the packages is not installed, but loaded (not attached) later on-demand via \texttt{requireNamespace()}.

Methods

Public methods:

- \texttt{ResultAssignerArchive$new()}
- \texttt{ResultAssignerArchive$assign_result()}
- \texttt{ResultAssignerArchive$clone()}

Method \texttt{new()}: Creates a new instance of this \texttt{R6} class.
Usage:
ResultAssignerArchive$new()

Method assign_result(): Assigns the result, i.e., the final point(s) to the instance.
Usage:
ResultAssignerArchive$assign_result(instance)
Arguments:
instance (bbotk::OptimInstanceSingleCrit | bbotk::OptimInstanceMultiCrit)
The bbotk::OptimInstance the final result should be assigned to.

Method clone(): The objects of this class are cloneable with this method.
Usage:
ResultAssignerArchive$clone(deep = FALSE)
Arguments:
deep Whether to make a deep clone.

See Also
Other Result Assigner: ResultAssigner, mlr_result_assigners, mlr_result_assigners_surrogate

Examples
result.assigner = ras("archive")

mland_result_assigners_surrogate

Result Assigner Based on a Surrogate Mean Prediction

Description
Result assigner that chooses the final point(s) based on a surrogate mean prediction of all evaluated
points in the bbotk::Archive. This is especially useful in the case of noisy objective functions.
In the case of operating on an bbotk::OptimInstanceMultiCrit the SurrogateLearnerCollection must
use as many learners as there are objective functions.

Super class
mland3mbo::ResultAssigner -> ResultAssignerSurrogate

Active bindings

surrogate (Surrogate | NULL)
The surrogate.
packages (character())
Set of required packages. A warning is signaled if at least one of the packages is not installed,
but loaded (not attached) later on-demand via requireNamespace().
Methods

Public methods:

- `ResultAssignerSurrogate$new()`
- `ResultAssignerSurrogate$assign_result()`
- `ResultAssignerSurrogate$clone()`

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

Usage:
`ResultAssignerSurrogate$new(surrogate = NULL)`

Arguments:
- `surrogate` (Surrogate | NULL)
  The surrogate that is used to predict the mean of all evaluated points.

Method `assign_result()`: Assigns the result, i.e., the final point(s) to the instance. If `surrogate` is NULL, `default_surrogate(instance)` is used and also assigned to `surrogate`.

Usage:
`ResultAssignerSurrogate$assign_result(instance)`

Arguments:
- `instance` (bbotk::OptimInstanceSingleCrit | bbotk::OptimInstanceMultiCrit)
  The bbotk::OptimInstance the final result should be assigned to.

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

Usage:
`ResultAssignerSurrogate$clone(deep = FALSE)`

Arguments:
- `deep` Whether to make a deep clone.

See Also

Other Result Assigner: `ResultAssigner`, `mlr_result_assigners`, `mlr_result_assigners_archive`

Examples

```r
result_assigner = ras("surrogate")
```
TunerMbo class that implements Model Based Optimization (MBO). This is a minimal interface internally passing on to OptimizerMbo. For additional information and documentation see OptimizerMbo.

Super classes

\texttt{mlr3tuning::Tuner} -> \texttt{mlr3tuning::TunerFromOptimizer} -> TunerMbo

Active bindings

- \texttt{loop\_function} (\texttt{loop\_function} | \texttt{NULL})
  - Loop function determining the MBO flavor.
- \texttt{surrogate} (\texttt{Surrogate} | \texttt{NULL})
  - The surrogate.
- \texttt{acq\_function} (\texttt{AcqFunction} | \texttt{NULL})
  - The acquisition function.
- \texttt{acq\_optimizer} (\texttt{AcqOptimizer} | \texttt{NULL})
  - The acquisition function optimizer.
- \texttt{args} (\texttt{named list()})
  - Further arguments passed to the \texttt{loop\_function}. For example, \texttt{random\_interleave\_iter}.
- \texttt{result\_assigner} (\texttt{ResultAssigner} | \texttt{NULL})
  - The result assigner.
- \texttt{param\_classes} (\texttt{character()})
  - Supported parameter classes that the optimizer can optimize. Determined based on the surrogate and the \texttt{acq\_optimizer}. This corresponds to the values given by a \texttt{paradox::ParamSet}'s \$class field.
- \texttt{properties} (\texttt{character()})
  - Set of properties of the optimizer. Must be a subset of \texttt{bbotk\_reflections$optimizer\_properties}. MBO in principle is very flexible and by default we assume that the optimizer has all properties. When fully initialized, properties are determined based on the \texttt{loop\_function} and \texttt{surrogate}.
- \texttt{packages} (\texttt{character()})
  - Set of required packages. A warning is signaled prior to optimization if at least one of the packages is not installed, but loaded (not attached) later on-demand via \texttt{requireNamespace()}. Required packages are determined based on the \texttt{acq\_function}, \texttt{surrogate} and the \texttt{acq\_optimizer}.
Methods

Public methods:

- TunerMbo$new()
- TunerMbo$print()
- TunerMbo$reset()
- TunerMbo$clone()

Method new(): Creates a new instance of this R6 class. For more information on default values for loop_function, surrogate, acq_function and acq_optimizer, see ?mbo_defaults. Note that all the parameters below are simply passed to the OptimizerMbo and the respective fields are simply (settable) active bindings to the fields of the OptimizerMbo.

Usage:

```r
TunerMbo$new(
  loop_function = NULL,
  surrogate = NULL,
  acq_function = NULL,
  acq_optimizer = NULL,
  args = NULL,
  result_assigner = NULL
)
```

Arguments:

- `loop_function` (loop_function | NULL): Loop function determining the MBO flavor.
- `surrogate` (Surrogate | NULL): The surrogate.
- `acq_function` (AcqFunction | NULL): The acquisition function.
- `acq_optimizer` (AcqOptimizer | NULL): The acquisition function optimizer.
- `args` (named list()): Further arguments passed to the loop_function. For example, random_interleave_iter.
- `result_assigner` (ResultAssigner | NULL): The result assigner.

Method print(): Print method.

Usage:

```r
TunerMbo$print()
```

Returns: (character()).

Method reset(): Reset the tuner. Sets the following fields to NULL: loop_function, surrogate, acq_function, acq_optimizer, args, result_assigner

Usage:

```r
TunerMbo$reset()
```

Method clone(): The objects of this class are cloneable with this method.
Usage:
TunerMbo$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.

Examples

if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {

  library(mlr3)
  library(mlr3tuning)

  # single-objective
  task = tsk("wine")
  learner = lrn("classif.rpart", cp = to_tune(lower = 1e-4, upper = 1, logscale = TRUE))
  resampling = rsmp("cv", folds = 3)
  measure = msr("classif.acc")

  instance = TuningInstanceSingleCrit$new(
    task = task,
    learner = learner,
    resampling = resampling,
    measure = measure,
    terminator = trm("evals", n_evals = 5))

  tnr("mbo")$optimize(instance)

  # multi-objective
  task = tsk("wine")
  learner = lrn("classif.rpart", cp = to_tune(lower = 1e-4, upper = 1, logscale = TRUE))
  resampling = rsmp("cv", folds = 3)
  measures = msrs(c("classif.acc", "selected_features"))

  instance = TuningInstanceMultiCrit$new(
    task = task,
    learner = learner,
    resampling = resampling,
    measures = measures,
    terminator = trm("evals", n_evals = 5),
    store_models = TRUE) # required due to selected features

  tnr("mbo")$optimize(instance)
}

ras

**Syntactic Sugar Result Assigner Construction**

**Description**

This function complements mlr_result_assigners with functions in the spirit of mlr_sugar from mlr3.

**Usage**

ras(.key, ...)

**Arguments**

- `.key` (character(1))
  Key passed to the respective dictionary to retrieve the object.
- `...` (named list())
  Named arguments passed to the constructor, to be set as parameters in the paradox::ParamSet, or to be set as public field. See mlr3misc::dictionary_sugar_get() for more details.

**Value**

ResultAssigner

**Examples**

ras("archive")

---

**ResultAssigner**

**Result Assigner Base Class**

**Description**

Abstract result assigner class.

A result assigner is responsible for assigning the final optimization result to the bbotk::OptimInstance. Normally, it is only used within an OptimizerMbo.

**Active bindings**

- `label` (character(1))
  Label for this object.
- `man` (character(1))
  String in the format [pkg]::[topic] pointing to a manual page for this object.
- `packages` (character())
  Set of required packages. A warning is signaled if at least one of the packages is not installed, but loaded (not attached) later on-demand via requireNamespace().
Methods

Public methods:

- `ResultAssigner$new()`
- `ResultAssigner$assign_result()`
- `ResultAssigner$format()`
- `ResultAssigner$print()`
- `ResultAssigner$clone()`

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

**Usage:**

```r
ResultAssigner$new(label = NA_character_, man = NA_character_)
```

**Arguments:**

- `label` (character(1))
  - Label for this object.
- `man` (character(1))
  - String in the format `[pkg]::[topic]` pointing to a manual page for this object.

Method `assign_result()`: Assigns the result, i.e., the final point(s) to the instance.

**Usage:**

```r
ResultAssigner$assign_result(instance)
```

**Arguments:**

- `instance` (bbotk::OptimInstanceSingleCrit | bbotk::OptimInstanceMultiCrit)
  - The bbotk::OptimInstance the final result should be assigned to.

Method `format()`: Helper for print outputs.

**Usage:**

```r
ResultAssigner$format()
```

Method `print()`: Print method.

**Usage:**

```r
ResultAssigner$print()
```

**Returns:** (character()).

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

**Usage:**

```r
ResultAssigner$clone(deep = FALSE)
```

**Arguments:**

- `deep` Whether to make a deep clone.

See Also

Other Result Assigner: `mlr_result_assigners`, `mlr_result_assigners_archive`, `mlr_result_assigners_surrogate`
**Description**

This function allows to construct a `SurrogateLearner` or `SurrogateLearnerCollection` in the spirit of `mlr_sugar` from `mlr3`.

If the archive references more than one target variable or `cols_y` contains more than one target variable but only a single learner is specified, this learner is replicated as many times as needed to build the `SurrogateLearnerCollection`.

**Usage**

```r
srlrn(learner, archive = NULL, cols_x = NULL, cols_y = NULL, ...)
```

**Arguments**

- `learner` *(mlr3::LearnerRegr | List of mlr3::LearnerRegr)*
  - `mlr3::LearnerRegr` that is to be used within the `SurrogateLearner` or a list of `mlr3::LearnerRegr` that are to be used within the `SurrogateLearnerCollection`.

- `archive` *(NULL | bbotk::Archive)*
  - `bbotk::Archive` of the `bbotk::OptimInstance` used. Can also be `NULL`.

- `cols_x` *(NULL | character())*
  - Column ids in the `bbotk::Archive` that should be used as features. Can also be `NULL` in which case this is automatically inferred based on the archive.

- `cols_y` *(NULL | character())*
  - Column id(s) in the `bbotk::Archive` that should be used as a target. If a list of `mlr3::LearnerRegr` is provided as the learner argument and `cols_y` is specified as well, as many column names as learners must be provided. Can also be `NULL` in which case this is automatically inferred based on the archive.

- `...` *(named list()*)
  - Named arguments passed to the constructor, to be set as parameters in the `paradox::ParamSet`.

**Value**

`SurrogateLearner` | `SurrogateLearnerCollection`

**Examples**

```r
library(mlr3)
srlrn(lrn("regr.featureless"), catch_errors = FALSE)
srlrn(list(lrn("regr.featureless"), lrn("regr.featureless")))
```
Surrogate Model

Description

Abstract surrogate model class.
A surrogate model is used to model the unknown objective function(s) based on all points evaluated so far.

Public fields

learner (learner)
Arbitrary learner object depending on the subclass.

Active bindings

print_id (character)
Id used when printing.
archive (bbotk::Archive | NULL)
   bbotk::Archive of the bbotk::OptimInstance.
n_learner (integer(1))
   Returns the number of surrogate models.
cols_x (character() | NULL)
   Column id’s of variables that should be used as features. By default, automatically inferred based on the archive.
cols_y (character() | NULL)
   Column id’s of variables that should be used as targets. By default, automatically inferred based on the archive.
insample_perf (numeric())
   Surrogate model’s current insample performance.
param_set (paradox::ParamSet)
   Set of hyperparameters.
assert_insample_perf (numeric())
   Asserts whether the current insample performance meets the performance threshold.
packages (character())
   Set of required packages. A warning is signaled if at least one of the packages is not installed, but loaded (not attached) later on-demand via requireNamespace().
feature_types (character())
   Stores the feature types the surrogate can handle, e.g. "logical", "numeric", or "factor". A complete list of candidate feature types, grouped by task type, is stored in mlr_reflections$task_feature_types.
properties (character())
   Stores a set of properties/capabilities the surrogate has. A complete list of candidate properties, grouped by task type, is stored in mlr_reflections$learner_properties.
predict_type (character(1))
   Retrieves the currently active predict type, e.g. "response".
Methods

Public methods:

- `Surrogate$new()`
- `Surrogate$update()`
- `Surrogate$predict()`
- `Surrogate$format()`
- `Surrogate$print()`
- `Surrogate$clone()`

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

Usage:
`Surrogate$new(learner, archive, cols_x, cols_y, param_set)`

Arguments:
  
  learner (learner)
  Arbitrary learner object depending on the subclass.

  archive (bbotk::Archive | NULL)
  bbotk::Archive of the bbotk::OptimInstance.

  cols_x (character() | NULL)
  Column id’s of variables that should be used as features. By default, automatically inferred based on the archive.

  cols_y (character() | NULL)
  Column id’s of variables that should be used as targets. By default, automatically inferred based on the archive.

  param_set (paradox::ParamSet)
  Parameter space description depending on the subclass.

Method `update()`: Train learner with new data. Subclasses must implement `$private.update()`.

Usage:
`Surrogate$update()`

Returns: NULL.

Method `predict()`: Predict mean response and standard error. Must be implemented by subclasses.

Usage:
`Surrogate$predict(xdt)`

Arguments:
  
  xdt (data.table::data.table())
  New data. One row per observation.

Returns: Arbitrary prediction object.

Method `format()`: Helper for print outputs.

Usage:
`Surrogate$format()`
**Method** `print()`: Print method.

*Usage:*  
`Surrogate$print()`  

*Returns:* (character(1)).

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

*Usage:*  
`Surrogate$clone(deep = FALSE)`  

*Arguments:*  
`deep` Whether to make a deep clone.

---

**Surrogate Learner**

**Surrogate Model Containing a Single Learner**

**Description**

Surrogate model containing a single `mlr3::LearnerRegr`.

**Parameters**

- `assert_insample_perf` logical(1)  
  Should the insample performance of the `mlr3::LearnerRegr` be asserted after updating the surrogate? If the assertion fails (i.e., the insample performance based on the `perf_measure` does not meet the `perf_threshold`), an error is thrown. Default is FALSE.

- `perf_measure` `mlr3::MeasureRegr`  
  Performance measure which should be use to assert the insample performance of the `mlr3::LearnerRegr`. Only relevant if `assert_insample_perf = TRUE`. Default is `mlr3::mlr_measures_regr.rsq`.

- `perf_threshold` numeric(1)  
  Threshold the insample performance of the `mlr3::LearnerRegr` should be asserted against. Only relevant if `assert_insample_perf = TRUE`. Default is 0.

- `catch_errors` logical(1)  
  Should errors during updating the surrogate be caught and propagated to the `loop_function` which can then handle the failed acquisition function optimization (as a result of the failed surrogate) appropriately by, e.g., proposing a randomly sampled point for evaluation? Default is TRUE.

**Super class**

`mlr3mbo::Surrogate` -> `Surrogate Learner`
**Active bindings**

- print_id (character)
  - Id used when printing.
- n_learner (integer(1))
  - Returns the number of surrogate models.
- assert_insample_perf (numeric())
  - Asserts whether the current insample performance meets the performance threshold.
- packages (character())
  - Set of required packages. A warning is signaled if at least one of the packages is not installed, but loaded (not attached) later on-demand via `requireNamespace()`.
- feature_types (character())
  - Stores the feature types the surrogate can handle, e.g. "logical", "numeric", or "factor". A complete list of candidate feature types, grouped by task type, is stored in `mlr_reflections$task_feature_types`.
- properties (character())
  - Stores a set of properties/capabilities the surrogate has. A complete list of candidate properties, grouped by task type, is stored in `mlr_reflections$learner_properties`.
- predict_type (character(1))
  - Retrieves the currently active predict type, e.g. "response".

**Methods**

**Public methods:**
- `SurrogateLearner$new()`
- `SurrogateLearner$predict()`
- `SurrogateLearner$clone()`

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

*Usage:*

```r
SurrogateLearner$new(learner, archive = NULL, cols_x = NULL, col_y = NULL)
```

*Arguments:*

- learner (mlr3::LearnerRegr).
- archive (bbotk::Archive | NULL)
  - bbotk::Archive of the bbotk::OptimInstance.
- cols_x (character() | NULL)
  - Column id’s of variables that should be used as features. By default, automatically inferred based on the archive.
- col_y (character(1) | NULL)
  - Column id of variable that should be used as a target. By default, automatically inferred based on the archive.

**Method predict():** Predict mean response and standard error.

*Usage:*

```r
SurrogateLearner$predict(xdt)
```

*Arguments:*


Surrogate Learner Collection

New data. One row per observation.

Returns: `data.table::data.table()` with the columns `mean` and `se`.

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

Usage:

```r
SurrogateLearner$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```r
if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud")) {
    library(bbotk)
    library(paradox)
    library(mlr3learners)

    fun = function(xs) {
        list(y = xs$x ^ 2)
    }
    domain = ps(x = p_dbl(lower = -10, upper = 10))
    codomain = ps(y = p_dbl(tags = "minimize"))
    objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

    instance = OptimInstanceSingleCrit$new(
        objective = objective,
        terminator = trm("evals", n_evals = 5))

    xdt = generate_design_random(instance$search_space, n = 4)$data

    instance$eval_batch(xdt)

    learner = default_gp()

    surrogate = srlrn(learner, archive = instance$archive)

    surrogate$update()

    surrogate$learner$model
}
```
**Description**

Surrogate model containing multiple `mlr3::LearnerRegr`. The `mlr3::LearnerRegr` are fit on the target variables as indicated via `cols_y`. Note that redundant `mlr3::LearnerRegr` must be deep clones.

**Parameters**

- `assert_insample_perf` logical(1)
  Should the insample performance of the `mlr3::LearnerRegr` be asserted after updating the surrogate? If the assertion fails (i.e., the insample performance based on the `perf_measure` does not meet the `perf_threshold`), an error is thrown. Default is `FALSE`.

- `perf_measure` List of `mlr3::MeasureRegr`
  Performance measures which should be use to assert the insample performance of the `mlr3::LearnerRegr`. Only relevant if `assert_insample_perf = TRUE`. Default is `mlr3::mlr_measures_regr.rsq` for each learner.

- `perf_threshold` List of numeric(1)
  Thresholds the insample performance of the `mlr3::LearnerRegr` should be asserted against. Only relevant if `assert_insample_perf = TRUE`. Default is `0` for each learner.

- `catch_errors` logical(1)
  Should errors during updating the surrogate be caught and propagated to the `loop_function` which can then handle the failed acquisition function optimization (as a result of the failed surrogate) appropriately by, e.g., proposing a randomly sampled point for evaluation? Default is `TRUE`.

**Super class**

`mlr3mbo::Surrogate` -> `SurrogateLearnerCollection`

**Active bindings**

- `print_id` (character)
  Id used when printing.

- `n_learner` (integer(1))
  Returns the number of surrogate models.

- `assert_insample_perf` (numeric())
  Asserts whether the current insample performance meets the performance threshold.

- `packages` (character())
  Set of required packages. A warning is signaled if at least one of the packages is not installed, but loaded (not attached) later on-demand via `requireNamespace()`.

- `feature_types` (character())
  Stores the feature types the surrogate can handle, e.g. "logical", "numeric", or "factor". A complete list of candidate feature types, grouped by task type, is stored in `mlr_reflections$task_feature_types`.

- `properties` (character())
  Stores a set of properties/capabilities the surrogate has. A complete list of candidate properties, grouped by task type, is stored in `mlr_reflections$learner_properties`.

- `predict_type` (character(1))
  Retrieves the currently active predict type, e.g. "response".
Methods

Public methods:

- `SurrogateLearnerCollection$new()`
- `SurrogateLearnerCollection$predict()`
- `SurrogateLearnerCollection$clone()`

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

Usage:
```
SurrogateLearnerCollection$new(
  learners,
  archive = NULL,
  cols_x = NULL,
  cols_y = NULL
)
```

Arguments:
- learners (list of `mlr3::LearnerRegr`).
- archive (bbotk::Archive | NULL)
  bbotk::Archive of the bbotk::OptimInstance.
- cols_x (character() | NULL)
  Column id’s of variables that should be used as features. By default, automatically inferred based on the archive.
- cols_y (character() | NULL)
  Column id’s of variables that should be used as targets. By default, automatically inferred based on the archive.


Usage:
```
SurrogateLearnerCollection$predict(xdt)
```

Arguments:
- xdt (data.table::data.table())
  New data. One row per observation.

Returns: list of data.table::data.table()s with the columns `mean` and `se`.

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

Usage:
```
SurrogateLearnerCollection$clone(deep = FALSE)
```

Arguments:
- deep Whether to make a deep clone.
Examples

if (requireNamespace("mlr3learners") &
    requireNamespace("DiceKriging") &
    requireNamespace("rgenoud") &
    requireNamespace("ranger")) {
library(bbotk)
library(paradox)
library(mlr3learners)

fun = function(xs) {
  list(y1 = xs$x^2, y2 = (xs$x - 2) ^ 2)
}
domain = ps(x = p_dbl(lower = -10, upper = 10))
codomain = ps(y1 = p_dbl(tags = "minimize"), y2 = p_dbl(tags = "minimize"))
optim_instance = OptimInstanceMultiCrit$new(fun = fun, domain = domain, codomain = codomain)

instance = optim_instance
objective = optim_instance$objective

xdt = generate_design_random(instance$search_space, n = 4)$data

learner1 = default_gp()
learner2 = default_rf()

surrogate = srlrn(list(learner1, learner2), archive = instance$archive)
surrogate$update()
surrogate$learner

surrogate$learner[["y2"]]
}
Index

* Acquisition Function
  AcqFunction, 5
  mlr_acqfunctions, 17
  mlr_acqfunctions_aei, 17
  mlr_acqfunctions_cb, 20
  mlr_acqfunctions_ehvi, 21
  mlr_acqfunctions_ehvigh, 23
  mlr_acqfunctions_ei, 26
  mlr_acqfunctions_eips, 28
  mlr_acqfunctions_mean, 30
  mlr_acqfunctions_pi, 32
  mlr_acqfunctions_sd, 33
  mlr_acqfunctions_smsego, 35

* Dictionary
  mlr_acqfunctions, 17
  mlr_loop_functions, 37
  mlr_result_assigners, 54

* Loop Function
  loop_function, 16
  mlr_loop_functions, 37
  mlr_loop_functions_ego, 38
  mlr_loop_functions_emo, 41
  mlr_loop_functions_mpcl, 43
  mlr_loop_functions_parego, 45
  mlr_loop_functions_smsego, 48

* Result Assigner
  mlr_result_assigners, 54
  mlr_result_assigners_archive, 55
  mlr_result_assigners_surrogate, 56
  ResultAssigner, 61

* datasets
  mlr_acqfunctions, 17
  mlr_loop_functions, 37
  mlr_result_assigners, 54

* mbo_defaults
  default_acqfunction, 11
  default_acqoptimizer, 11
  default_gp, 12
  default_loop_function, 13
  default_result_assigner, 13
  default_rf, 14
  default_surrogate, 14
  mbo_defaults, 16
  acqf, 4
  acqf(), 17, 20, 26, 28, 30, 32, 34
  AcqFunction, 5, 5, 8, 9, 11, 12, 17, 19–21, 23,
  25–34, 37, 39, 41, 44, 46, 51–53, 58,
  59
  AcqFunctionAEI (mlr_acqfunctions_aei), 17
  AcqFunctionCB (mlr_acqfunctions_cb), 20
  AcqFunctionEHVI, 24
  AcqFunctionEHVI
    (mlr_acqfunctions_ehvi), 21
  AcqFunctionEHVIGH, 22
  AcqFunctionEHVIGH
    (mlr_acqfunctions_ehvigh), 23
  AcqFunctionEI (mlr_acqfunctions_ei), 26
  AcqFunctionEIPS
    (mlr_acqfunctions_eips), 28
  AcqFunctionMean
    (mlr_acqfunctions_mean), 30
  AcqFunctionPI (mlr_acqfunctions_pi), 32
  AcqFunctionSD (mlr_acqfunctions_sd), 33
  AcqFunctionSmsego
    (mlr_acqfunctions_smsego), 35
  acqo, 7
  AcqOptimizer, 7, 8, 8, 12, 39, 41, 44, 46, 49,
  51, 53, 58, 59

  bayesopt_ego, 13, 16, 51
  bayesopt_ego (mlr_loop_functions_ego), 38
  bayesopt_emo (mlr_loop_functions_emo), 41
  bayesopt_mpcl
    (mlr_loop_functions_mpcl), 43

  72
bayesopt::parego
  (mlr::loop_functions::parego), 45
bayesopt::parego(), 51
bayesopt::smsego, 13
bayesopt::smsego
  (mlr::loop_functions::smsego), 48
bbotk::Archive, 5, 8, 9, 39, 41, 42, 44, 46, 
  47, 49, 51, 52, 55, 56, 63–65, 67, 70
bbotk::Objective, 5, 18, 20, 22, 24, 26, 28, 
  30, 32, 34, 35
bbotk::OptimInstance, 5, 6, 8, 11, 13, 15, 
  36, 51, 52, 56, 57, 61–65, 67, 70
bbotk::OptimInstanceMultiCrit, 15, 41, 
  42, 46, 47, 49, 56, 57, 62
bbotk::OptimInstanceSingleCrit, 15, 28, 
  39, 43, 44, 56, 57, 62
bbotk::Optimizer, 8, 9, 51, 55
bbotk::OptimizerRandomSearch, 11
bbotk::Terminator, 8, 9, 49, 51
bbotk::TerminatorEvals, 11, 36, 49
bbotk::reflection::optimizer_properties, 
  52, 58
data.table::data.table(), 7, 10, 65, 68, 
  70
default_acqfunction, 11, 12–14, 16
default_acqoptimizer, 11, 11, 12–14, 16
default_gp, 11, 12, 12, 13, 14, 16
default_gp(), 15
default_loop_function, 11, 12, 13, 14, 16
default_result_assigner, 11–13, 13, 14, 
  16
default_rf, 11–14, 14, 16
default_rf(), 15
default_surrogate, 11–14, 14, 16
dictionary, 4, 17, 20, 26, 28, 30, 32, 34, 61
fastGHQuad::gaussHermiteData, 24
loop_function, 13, 16, 38, 39, 42, 45, 47, 
  50–52, 58, 59
mbo_defaults, 11–14, 16
mlr::Learner, 15
mlr::LearnerRegr, 12, 14, 63, 66, 67, 69, 70
mlr::MeasureRegr, 66, 69
mlr::mlr_measures::regr::rsq, 66, 69
mlr3learners, 18
mlr3mbo (mlr3mbo-package), 3
mlr3mbo::AcqFunction, 18, 20, 22, 24, 26, 
  28, 30, 32, 34, 35
mlr3mbo::ResultAssigner, 55, 56
mlr3mbo::Surrogate, 66, 69
mlr3misc::Dictionary, 17, 37, 38, 54, 55
mlr3misc::dictionary::sugar::get(), 4, 61
mlr3tuning::Tuner, 58
mlr3tuning::TunerFromOptimizer, 58
mlr_acqfunctions::cb, 4, 7, 17, 17, 19–21, 23, 
  25–34, 37, 38, 55
mlr_acqfunctions::aei, 7, 17, 17, 21, 23, 25, 
  27, 29, 31, 33, 34, 37
mlr_acqfunctions::cb, 7, 17, 19, 20, 23, 25, 
  27, 29, 31, 33, 34, 37
mlr_acqfunctions::ehvi, 7, 17, 17, 19, 21, 23, 
  25, 29, 31, 33, 34, 37
mlr_acqfunctions::ehvigh, 7, 17, 19, 21, 23, 
  23, 27, 29, 31, 33, 34, 37
mlr_acqfunctions::ei, 7, 11, 17, 19, 21, 23, 
  25, 29, 31, 33, 34, 37, 51
mlr_acqfunctions::eips, 7, 17, 19, 21, 23, 
  25, 27, 28, 31, 33, 34, 37
mlr_acqfunctions::mean, 7, 17, 19, 21, 23, 
  25, 27, 29, 30, 33, 34, 37
mlr_acqfunctions::pi, 7, 17, 19, 21, 23, 25, 
  27, 29, 31, 32, 34, 37
mlr_acqfunctions::sd, 7, 17, 19, 21, 23, 25, 
  27, 29, 31, 33, 33, 37
mlr_acqfunctions::smsego, 7, 11, 17, 19, 21, 
  23, 25, 27, 29, 31, 33, 34, 35, 48, 49
mlr_loop_functions::cb, 16, 17, 37, 39, 42, 45, 
  47, 50, 55
mlr_loop_functions::ego, 16, 38, 39, 41, 42, 
  45, 47, 50
mlr_loop_functions::emo, 16, 38, 39, 41, 45, 
  47, 50
mlr_loop_functions::mpcl, 16, 38, 39, 42, 
  43, 47, 50
mlr_loop_functions::parego, 16, 38, 39, 42, 
  43, 45, 50
mlr_loop_functions::smsego, 16, 38, 39, 42, 
  45, 47, 48
mlr_optimizers::mbo, 50
mlr::reflection::learner_properties, 64, 
  67, 69
mlr::reflection::task::feature.types, 64, 
  67, 69
mlr_result_assigners, 17, 38, 54, 56, 57, 61, 62
mlr_result_assigners_archive, 55, 55, 57, 62
mlr_result_assigners_surrogate, 55, 56, 56, 62
mlr_tuners_mbo, 58

OptimizerMbo, 16, 38, 41, 43, 46, 48, 58, 59, 61
OptimizerMbo (mlr_optimizers_mbo), 50

paradox::ParamSet, 4, 6, 8, 9, 52, 58, 61, 63–65

R6, 6, 9, 18, 20, 22, 24, 26, 29, 30, 32, 34, 36, 52, 55, 57, 59, 62, 65, 67, 70
R6::R6Class, 17, 38, 55

ras, 61
ras(), 54, 55
requireNamespace(), 6, 52, 55, 56, 58, 61, 64, 67, 69

ResultAssigner, 13, 52–59, 61, 61
ResultAssignerArchive, 13
ResultAssignerArchive
  (mlr_result_assigners_archive), 55
ResultAssignerSurrogate
  (mlr_result_assigners_surrogate), 56

srlrn, 63

Surrogate, 5, 6, 15, 39, 44, 51–53, 56–59, 64
SurrogateLearner, 15, 18, 20, 27, 31, 32, 34, 39, 44, 46, 63, 66
SurrogateLearnerCollection, 15, 22, 25, 28, 29, 36, 41, 49, 56, 63, 68

TunerMbo (mlr_tuners_mbo), 58