# Package ‘mlr3mbo’

November 18, 2022

**Type** Package

**Title** Flexible Bayesian Optimization

**Version** 0.1.1

**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:10.1007/978-3-540-87700-4_78>.

**License** LGPL-3

**URL** `https://mlr3mbo.mlr-org.com`, `https://github.com/mlr-org/mlr3mbo`

**BugReports** `https://github.com/mlr-org/mlr3mbo/issues`

**Depends** mlr3tuning (>= 0.14.0), 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), paradox (>= 0.10.0), R6 (>= 2.4.1)

**Suggests** DiceKriging, knitr, lhs, mlr3learners (>= 0.5.4), mlr3pipelines (>= 0.4.2), nloptr, ranger, rgenoud, rmarkdown, rpart, spacefillr, stringi, testthat (>= 3.0.0),

**ByteCompile** no

**Encoding** UTF-8

**Config/testthat/edition** 3

**Config/testthat/parallel** false

**NeedsCompilation** yes

**RoxygenNote** 7.2.1

**Collate** 'mlr_acqfunctions.R' 'AcqFunction.R' 'AcqFunctionCB.R' 'AcqFunctionEI.R' 'AcqFunctionEIPS.R' 'AcqFunctionMean.R' 'AcqFunctionPI.R' 'AcqFunctionSmsEgo.R' 'AcqOptimizer.R' 'OptimizerMbo.R' 'Surrogate.R' 'SurrogateLearner.R' 'SurrogateLearnerCollection.R' 'TunerMbo.R' 'mlr_loop_functions.R' 'bayesopt_ego.R' 'bayesopt_mpcl.R' 'bayesopt_parego.R' 'bayesopt_smsego.R' 'bibentries.R' 'helper.R' 'loop_function.R' 'mbo_defaults.R' 'result_by_default.R' 'result_by_surrogate_design.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** 2022-11-18 10:40:02 UTC

## R topics documented:

mlr3mbo-package . . . . . 3
acqf . . . . . 4
AcqFunction . . . . . 5
acqo . . . . . 7
AcqOptimizer . . . . . 8
default_acqfun . . . . . 10
default_acqopt . . . . . 10
default_loopfun . . . . . 11
default_surrogate . . . . . 11
loop_function . . . . . 13
mbo_defaults . . . . . 13
mlr_acqfunctions . . . . . 14
mlr_acqfunctions_cb . . . . . 14
mlr_acqfunctions_ei . . . . . 16
mlr_acqfunctions_eips . . . . . 18
mlr_acqfunctions_mean . . . . . 20

mlr_acqfunctions_pi . . . . . . . . . . 22
mlr_acqfunctions_smsego . . . . . . . . . . 24
mlr_loop_functions . . . . . . . . . . 26
mlr_loop_functions_ego . . . . . . . . . . 27
mlr_loop_functions_mpcl . . . . . . . . . . 30
mlr_loop_functions_parego . . . . . . . . . . 32
mlr_loop_functions_smsego . . . . . . . . . . 35
mlr_optimizers_mbo . . . . . . . . . . 37
mlr_tuners_mbo . . . . . . . . . . 41
result_by_default . . . . . . . . . . 44
result_by_surrogate_design . . . . . . . . . . 45
srlrn . . . . . . . . . . 45
srlrnc . . . . . . . . . . 46
Surrogate . . . . . . . . . . 47
SurrogateLearner . . . . . . . . . . 49
SurrogateLearnerCollection . . . . . . . . . . 52

**Index** **55**

---

`mlr3mbo-package` *mlr3mbo: Flexible Bayesian Optimization*

---

## 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)`)<br>Key passed to the respective dictionary to retrieve the object. |
| `...` | (named `list()`)<br>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

AcqFunction

## Examples

```
acqf("ei")
```

## `AcqFunction` *Acquisition Function Base Class*

### 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`)
: Pointing to the private acquistion function to be implemented by subclasses.

`surrogate` (Surrogate)
: Surrogate.

### 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.

Note that the surrogate can be initialized lazy and can later be set via the active binding `$surrogate`.

*Usage:*

```
AcqFunction$new(
  id,
  constants = ParamSet$new(),
  surrogate,
  direction,
  label = NA_character_,
  man = NA_character_
)
```

*Arguments:*

`id` (`character(1)`).

`constants` (paradox::ParamSet). Changeable constants or parameters.

`surrogate` (`NULL` | Surrogate). Surrogate whose predictions are used in the acquisition function.

`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"`.

`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:*

`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_cb`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions`

---

`acqo` *Syntactic Sugar Acquisition Function Optimizer Construction*

---

## Description

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

## Usage

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

## Arguments

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

## Value

AcqOptimizer

## Examples

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

`AcqOptimizer` *Acquisition Function Optimizer*

## Description

Optimizer for AcqFunctions which performs the infill optimization. Wraps an bbotk::Optimizer and bbotk::Terminator.

## Parameters

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

`warmstart logical(1)`
: Should the infill 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 infill optimizer, e.g., local search or mutation. Default is `FALSE`.

`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 infill 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 infill optimization be caught and propagated to the `loop_function` which can then handle the failed infill 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$print()`
- `AcqOptimizer$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:*

`deep` Whether to make a deep clone.

## `default_acqfun` *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

```
default_acqfun(instance)
```

### Arguments

| | |
|---|---|
| `instance` | (bbotk::OptimInstance). |

### Value

AcqFunction

### See Also

Other mbo_defaults: `default_acqopt()`, `default_loopfun()`, `default_surrogate()`, `mbo_defaults`

## `default_acqopt` *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

```
default_acqopt(acq_function)
```

### Arguments

| | |
|---|---|
| `acq_function` | (AcqFunction). |

### Value

AcqOptimizer

### See Also

Other mbo_defaults: `default_acqfun()`, `default_loopfun()`, `default_surrogate()`, `mbo_defaults`

## `default_loopfun` *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_loopfun(instance)
```

### Arguments

| | |
|---|---|
| `instance` | (bbotk::OptimInstance)<br>An object that inherits from bbotk::OptimInstance. |

### Value

loop_function

### See Also

Other mbo_defaults: `default_acqfun()`, `default_acqopt()`, `default_surrogate()`, `mbo_defaults`

## `default_surrogate` *Default Surrogate*

### Description

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

For numeric-only (including integers) parameter spaces without any dependencies:

- A Kriging model “"regr.km"” with kernel “"matern3_2"” is created.
- If the objective function is deterministic we add a small nugget effect (10^-8*Var(y), y is vector of observed outcomes in current design) to increase numerical stability to hopefully prevent crashes of **DiceKriging**. Whether the objective function is deterministic can be observed from the objective function’s properties.
- If the objective function is noisy the nugget effect will be estimated with `nugget.estim = TRUE`.
- Also `jitter` is set to `TRUE` to circumvent a problem with **DiceKriging** where already trained input values produce the exact trained output. Whether the objective function is noisy can be observed from the objective functions properties.

- 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 surrogate model.

For mixed numeric-categorical parameter spaces, or spaces with conditional parameters:

- A ranger regression forest “"regr.ranger"” with 500 trees is created.
- The standard error of a prediction (if required by the infill criterion) is estimated via jackknife. This is the `se.method = "jack"` option of the “"regr.ranger"” learner (default).

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 = 20L, 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 ranger random forest, more concretely we use `po("imputesample")` (for logicals) and `po("imputeoor")` (for anything else) from package **mlr3pipelines**. 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 deep clones of the learner are wrapped as a SurrogateLearnerCollection.

## Usage

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

## Arguments

| | |
|---|---|
| `instance` | (bbotk::OptimInstance)<br>An object that inherits from bbotk::OptimInstance. |
| `learner` | (`NULL` \| mlr3::Learner). |
| `n_learner` | (`NULL` \| `integer(1)`). |

## Value

Surrogate

## References

- Ding, Yufeng, Simonoff, S J (2010). “An investigation of missing data methods for classification trees applied to binary response data.” *Journal of Machine Learning Research*, **11**(1), 131–170.

## See Also

Other mbo_defaults: `default_acqfun()`, `default_acqopt()`, `default_loopfun()`, `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_ego`, `mlr_loop_functions_mpcl`, `mlr_loop_functions_parego`, `mlr_loop_functions_smsego`, `mlr_loop_functions`

## `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_loopfun
- Surrogate: default_surrogate
- Acquisition Function: default_acqfun
- Acqfun Optimizer: default_acqopt

### See Also

Other mbo_defaults: `default_acqfun()`, `default_acqopt()`, `default_loopfun()`, `default_surrogate()`

## mlr_acqfunctions *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`

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_smsego`

### Examples

```
as.data.table(mlr_acqfunctions)
acqf("ei")
```

## mlr_acqfunctions_cb *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()`:

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

## Parameters

- `"lambda"` (`numeric(1)`)
  $\lambda$ 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:*

```
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:*

```
AcqFunctionCB$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## References

- Snoek, Jasper, Larochelle, Hugo, Adams, P R (2012). "Practical Bayesian Optimization of Machine Learning Algorithms." In Pereira F, Burges CJC, Bottou L, Weinberger KQ (eds.), *Advances in Neural Information Processing Systems*, volume 25, 2951–2959.

## See Also

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions`

## 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 = lrn("regr.km",
    covtype = "matern3_2",
    optim.method = "gen",
    nugget.stability = 10^-8,
    control = list(trace = FALSE))

  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)))
}
```

## 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.

## 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:*

`deep` Whether to make a deep clone.

## References

- Jones, R. D, Schonlau, Matthias, Welch, J. W (1998). "Efficient Global Optimization of Expensive Black-Box Functions." *Journal of Global optimization*, **13**(4), 455–492.

## See Also

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions`

## 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 = lrn("regr.km",
    covtype = "matern3_2",
    optim.method = "gen",
    nugget.stability = 10^-8,
    control = list(trace = FALSE))

  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 `y_cols` 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()`:

```
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.

## Active bindings

`y_col` (`character(1)`).

`time_col` (`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:*

`deep` Whether to make a deep clone.

## References

- Snoek, Jasper, Larochelle, Hugo, Adams, P R (2012). "Practical Bayesian Optimization of Machine Learning Algorithms." In Pereira F, Burges CJC, Bottou L, Weinberger KQ (eds.), *Advances in Neural Information Processing Systems*, volume 25, 2951–2959.

## See Also

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions`

## Examples

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

  fun = function(xs) {
    list(y = xs$x ^ 2, time = abs(xs$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 = lrn("regr.km",
    covtype = "matern3_2",
    optim.method = "gen",
    nugget.stability = 10^-8,
    control = list(trace = FALSE))

 surrogate = srlrnc(list(learner, learner$clone(deep = TRUE)), archive = instance$archive)
  surrogate$y_cols = 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_cb`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions`

## 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 = lrn("regr.km",
    covtype = "matern3_2",
    optim.method = "gen",
    nugget.stability = 10^-8,
    control = list(trace = FALSE))

  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()`:

```
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.

### Methods

**Public methods:**

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

- AcqFunctionPI$clone()

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

*Usage:*

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

*Arguments:*

`surrogate` (NULL | SurrogateLearner).

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

*Usage:*

```
AcqFunctionPI$update()
```

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## References

- Kushner, J. H (1964). “A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise.” *Journal of Basic Engineering*, **86**(1), 97–106.

## See Also

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions`

## 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 = lrn("regr.km",
    covtype = "matern3_2",
    optim.method = "gen",
    nugget.stability = 10^-8,
    control = list(trace = FALSE))

  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_smsego

*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.

`ref_point` (`numeric()`)
  Reference point.

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

`progress` (`numeric(1)`)
    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:*

`deep` Whether to make a deep clone.

## References

- Ponweiser, Wolfgang, Wagner, Tobias, Biermann, Dirk, Vincze, Markus (2008). “Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted S-Metric Selection.” In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature*, 784–794.
- Horn, Daniel, Wagner, Tobias, Biermann, Dirk, Weihs, Claus, Bischl, Bernd (2015). “Model-Based Multi-objective Optimization: Taxonomy, Multi-Point Proposal, Toolbox and Benchmark.” In *International Conference on Evolutionary Multi-Criterion Optimization*, 64–78.

## See Also

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_pi`, `mlr_acqfunctions`

## Examples

```
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 = lrn("regr.km",
    covtype = "matern3_2",
    optim.method = "gen",
    nugget.stability = 10^-8,
    control = list(trace = FALSE))

  surrogate = srlrnc(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 *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.

## Arguments

| | |
|---|---|
| key | (character(1)). |
| ... | (any)<br>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`

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

## Examples

```
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,
  init_design_size = NULL,
  surrogate,
  acq_function,
  acq_optimizer,
  random_interleave_iter = 0L
)
```

## Arguments

| | |
|---|---|
| `instance` | (bbotk::OptimInstanceSingleCrit)<br>The bbotk::OptimInstanceSingleCrit to be optimized. |
| `init_design_size` | (`NULL` \| `integer(1)`)<br>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 drawn uniformly at random. |
| `surrogate` | (Surrogate)<br>Surrogate to be used as a surrogate. Typically a SurrogateLearner. |
| `acq_function` | (AcqFunction)<br>AcqFunction to be used as acquisition function. |
| `acq_optimizer` | (AcqOptimizer)<br>AcqOptimizer to be used as acquisition function optimizer. |
| `random_interleave_iter` | (`integer(1)`)<br>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

- Jones, R. D, Schonlau, Matthias, Welch, J. W (1998). “Efficient Global Optimization of Expensive Black-Box Functions.” *Journal of Global optimization*, **13**(4), 455–492.
- Snoek, Jasper, Larochelle, Hugo, Adams, P R (2012). “Practical Bayesian Optimization of Machine Learning Algorithms.” In Pereira F, Burges CJC, Bottou L, Weinberger KQ (eds.), *Advances in Neural Information Processing Systems*, volume 25, 2951–2959.

## See Also

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

## 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"),
    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$y_cols = c("y", "time")
```

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

    optimizer$optimize(instance)
  }
```

## mlr_loop_functions_mpcl

*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

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

### Arguments

| | |
|---|---|
| `instance` | (bbotk::OptimInstanceSingleCrit)<br>The bbotk::OptimInstanceSingleCrit to be optimized. |
| `init_design_size` | (`NULL` \| `integer(1)`)<br>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 drawn uniformly at random. |

| | |
|---|---|
| `surrogate` | (Surrogate)<br>Surrogate to be used as a surrogate. Typically a SurrogateLearner. |
| `acq_function` | (AcqFunction)<br>AcqFunction to be used as acquisition function. |
| `acq_optimizer` | (AcqOptimizer)<br>AcqOptimizer to be used as acquisition function optimizer. |
| `q` | (`integer(1)`)<br>Batch size > `1`. Default is `2`. |
| `liar` | (`function`)<br>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)`)<br>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

- Ginsbourger, David, Le Riche, Rodolphe, Carraro, Laurent (2008). “A Multi-Points Criterion for Deterministic Parallel Global Optimization Based on Gaussian processes.”
- Wang, Jialei, Clark, C. S, Liu, Eric, Frazier, I. P (2020). “Parallel Bayesian Global Optimization of Expensive Functions.” *Operations Research*, **68**(6), 1850–1865.

## See Also

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

## 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 = 7))

  surrogate = default_surrogate(instance)

  acq_function = acqf("ei")

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

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

  optimizer$optimize(instance)
}
```

---

## mlr_loop_functions_parego
*Multi-Objective Bayesian Optimization via ParEGO*

---

## 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,
  init_design_size = NULL,
  surrogate,
  acq_function,
  acq_optimizer,
  q = 1L,
  s = 100L,
  rho = 0.05,
  random_interleave_iter = 0L
)
```

## Arguments

| | |
|---|---|
| `instance` | (bbotk::OptimInstanceMultiCrit)<br>The bbotk::OptimInstanceMultiCrit to be optimized. |
| `init_design_size` | (`NULL` \| `integer(1)`)<br>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 drawn uniformly at random. |
| `surrogate` | (SurrogateLearner)<br>SurrogateLearner to be used as a surrogate. |
| `acq_function` | (AcqFunction)<br>AcqFunction to be used as acquisition function. |
| `acq_optimizer` | (AcqOptimizer)<br>AcqOptimizer to be used as acquisition function optimizer. |
| `q` | (`integer(1)`)<br>Batch size, i.e., the number of candidates to be obtained for a single batch. Default is `1`. |
| `s` | (`integer(1)`)<br>$s$ in Equation 1 in Knowles (2006). Determines the total number of possible random weight vectors. Default is `100`. |
| `rho` | (`numeric(1)`)<br>$\rho$ in Equation 2 in Knowles (2006) scaling the linear part of the augmented Tchebycheff function. Default is `0.05` |
| `random_interleave_iter` | (`integer(1)`)<br>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

- Knowles, Joshua (2006). “ParEGO: A Hybrid Algorithm With On-Line Landscape Approximation for Expensive Multiobjective Optimization Problems.” *IEEE Transactions on Evolutionary Computation*, **10**(1), 50–66.

## See Also

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

## 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, n_learner = 1)

    acq_function = acqf("ei")

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

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

    optimizer$optimize(instance)
  }
```

---

## `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,
  init_design_size = NULL,
  surrogate,
  acq_function,
  acq_optimizer,
  random_interleave_iter = 0L
)
```

### Arguments

| | |
|---|---|
| `instance` | (bbotk::OptimInstanceMultiCrit)<br>The bbotk::OptimInstanceMultiCrit to be optimized. |

| | |
|---|---|
| `init_design_size` | (`NULL` \| `integer(1)`)<br>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 drawn uniformly at random. |
| `surrogate` | (SurrogateLearnerCollection)<br>SurrogateLearnerCollection to be used as a surrogate. |
| `acq_function` | (mlr_acqfunctions_smsego)<br>mlr_acqfunctions_smsego to be used as acquisition function. |
| `acq_optimizer` | (AcqOptimizer)<br>AcqOptimizer to be used as acquisition function optimizer. |
| `random_interleave_iter` | (`integer(1)`)<br>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

- Beume N, Naujoks B, Emmerich M (2007). “SMS-EMOA: Multiobjective selection based on dominated hypervolume.” *European Journal of Operational Research*, **181**(3), 1653–1669.
- Ponweiser, Wolfgang, Wagner, Tobias, Biermann, Dirk, Vincze, Markus (2008). “Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted S-Metric Selection.” In *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature*, 784–794.

## See Also

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

## 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("smsego")

  acq_optimizer = acqo(
    optimizer = opt("random_search"),
    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)
}
```

---

## `mlr_optimizers_mbo` *Model Based Optimization*

---

## Description

`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.

## 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_function` (`function` | `NULL`)
  Optional function called after the optimization terminates. Determines how the final result of the optimization is calculated. Requires arguments `inst` (the bbotk::OptimInstance) and `self` (the OptimizerMbo). See for example result_by_surrogate_design which is used by default if the bbotk::OptimInstance has the property `"noisy"` (which is the case for a mlr3tuning::TuningInstanceSingleCrit or mlr3tuning::TuningInstanceMultiCrit).

`param_classes` (`character()`)
  Supported parameter classes that the optimizer can optimize. Determined based on the `surrogate` and the `acq_optimizer`. Subclasses of paradox::Param.

`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 `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_acqfun(instance)` is used. If `acq_optimizer` is NULL, `default_acqopt(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 `acq_optimizer`, see `?mbo_defaults`.

*Usage:*

```
OptimizerMbo$new(
  loop_function = NULL,
  surrogate = NULL,
  acq_function = NULL,
  acq_optimizer = NULL,
  args = NULL,
  result_function = 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_function` (`function | NULL`)
: Optional function called after the optimization terminates. Determines how the final result of the optimization is calculated. Requires arguments `inst` (the bbotk::OptimInstance)

and `self` (the OptimizerMbo). See for example result_by_surrogate_design which is used by default if the bbotk::OptimInstance has the property `"noisy"` (which is the case for a mlr3tuning::TuningInstanceSingleCrit or mlr3tuning::TuningInstanceMultiCrit).

**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_function`

*Usage:*

```
OptimizerMbo$reset()
```

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

*Usage:*

```
OptimizerMbo$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
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))

  learner = lrn("regr.km",
    covtype = "matern3_2",
    optim.method = "gen",
    nugget.stability = 10^-8,
    control = list(trace = FALSE))
  surrogate = srlrn(learner)

  acq_function = acqf("ei")

  acq_optimizer = acqo(
    optimizer = opt("random_search"),
    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)
}
```

## `mlr_tuners_mbo` *Tuner using Model Based Optimization*

### Description

`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

`mlr3tuning::Tuner` -> `mlr3tuning::TunerFromOptimizer` -> `TunerMbo`

## 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_function` (`function` | `NULL`)
: Optional function called after the optimization terminates. Determines how the final result of the optimization is calculated. Requires arguments `inst` (the bbotk::OptimInstance) and `self` (the OptimizerMbo). See for example result_by_surrogate_design which is used by default if the bbotk::OptimInstance has the property `"noisy"` (which is the case for a mlr3tuning::TuningInstanceSingleCrit or mlr3tuning::TuningInstanceMultiCrit).

`param_classes` (`character()`)
: Supported parameter classes that the optimizer can optimize. Determined based on the `surrogate` and the `acq_optimizer`. Subclasses of paradox::Param.

`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 `surrogate` and the `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:*

```
TunerMbo$new(
  loop_function = NULL,
  surrogate = NULL,
  acq_function = NULL,
  acq_optimizer = NULL,
  args = NULL,
  result_function = 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_function` (`function` | `NULL`)
Optional function called after the optimization terminates. Determines how the final result of the optimization is calculated. Requires arguments `inst` (the bbotk::OptimInstance) and `self` (the OptimizerMbo). See for example result_by_surrogate_design which is used by default if the bbotk::OptimInstance has the property `"noisy"` (which is the case for a mlr3tuning::TuningInstanceSingleCrit or mlr3tuning::TuningInstanceMultiCrit).

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

*Usage:*

```
TunerMbo$print()
```

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

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

*Usage:*

```
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)
}
```

---

# `result_by_default` *Choose Final Point(s) based on the Archive*

---

## Description

Choose final point(s) based on all evaluations in the archive.

## Usage

```
result_by_default(instance, optimizer_mbo)
```

## Arguments

| | |
|---|---|
| `instance` | (bbotk::OptimInstanceSingleCrit \| bbotk::OptimInstanceMultiCrit)<br>The bbotk::OptimInstance the result should be assigned to. |
| `optimizer_mbo` | (OptimizerMbo)<br>The OptimizerMbo that generates the final result. |

---

## `result_by_surrogate_design` *Choose Final Point(s) by Surrogate Mean*

---

### Description

Choose final point(s) by best surrogate mean prediction on all evaluated points.

### Usage

```
result_by_surrogate_design(instance, optimizer_mbo)
```

### Arguments

| | |
|---|---|
| `instance` | (bbotk::OptimInstanceSingleCrit \| bbotk::OptimInstanceMultiCrit)<br>The bbotk::OptimInstance the result should be assigned to. |
| `optimizer_mbo` | (OptimizerMbo)<br>The OptimizerMbo that generates the final result. |

---

## `srlrn` *Syntactic Sugar Surrogate Learner Construction*

---

### Description

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

### Usage

```
srlrn(learner, archive = NULL, x_cols = NULL, y_col = NULL, ...)
```

### Arguments

| | |
|---|---|
| `learner` | (mlr3::LearnerRegr)<br>mlr3::LearnerRegr that is to be used. |
| `archive` | (`NULL` \| bbotk::Archive)<br>bbotk::Archive of the bbotk::OptimInstance used. Can also be `NULL`. |
| `x_cols` | (`NULL` \| `character()`)<br>Names of columns in the bbotk::Archive that should be used as features. Can also be `NULL`. |
| `y_col` | (`NULL` \| `character(1)`)<br>Name of the column in the bbotk::Archive that should be used as a target. Can also be `NULL`. |
| `...` | (named `list()`)<br>Named arguments passed to the constructor, to be set as parameters in the paradox::ParamSet. |

### Value

SurrogateLearner

### Examples

```
srlrn(lrn("regr.featureless"), catch_errors = FALSE)
```

## srlrnc *Syntactic Sugar Surrogate Learner Collection Construction*

### Description

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

### Usage

```
srlrnc(learners, archive = NULL, x_cols = NULL, y_cols = NULL, ...)
```

### Arguments

| | |
|---|---|
| `learners` | (List of mlr3::LearnerRegr)<br>mlr3::LearnerRegr that are to be used. |
| `archive` | (`NULL` \| bbotk::Archive)<br>bbotk::Archive of the bbotk::OptimInstance used. Can also be `NULL`. |
| `x_cols` | (`NULL` \| `character()`)<br>Names of columns in the bbotk::Archive that should be used as features. Can also be `NULL`. |

| | |
|---|---|
| `y_cols` | (`NULL` \| `character()`)<br>Names of the columns in the bbotk::Archive that should be used as targets. Can also be `NULL`. |
| `...` | (named `list()`)<br>Named arguments passed to the constructor, to be set as parameters in the paradox::ParamSet. |

## Value

SurrogateLearnerCollection

## Examples

```
srlrnc(list(lrn("regr.featureless"), lrn("regr.featureless")), catch_errors = FALSE)
```

---

`Surrogate` *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

`model` (model)
: Arbitrary model 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.

`x_cols` (`character()` | `NULL`)
: Column Id's of variables that should be used as features. By default, automatically inferred based on the archive.

`y_cols` (`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`.

## 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(model, archive, x_cols, y_cols, param_set)
```

*Arguments:*

`model` (model)
Arbitrary model object depending on the subclass.

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

`x_cols` (`character()` | NULL)
Column Id's of variables that should be used as features. By default, automatically inferred based on the archive.

`y_cols` (`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 model 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()`).

**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.

---

## `SurrogateLearner` *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 infill 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` -> `SurrogateLearner`

## 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 learner has. A complete list of candidate properties, grouped by task type, is stored in `mlr_reflections$learner_properties`.

## Methods

### Public methods:

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

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

*Usage:*

```
SurrogateLearner$new(learner, archive = NULL, x_cols = NULL, y_col = NULL)
```

*Arguments:*

`learner` (mlr3::LearnerRegr).

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

`x_cols` (`character()` | `NULL`)
Column Id's of variables that should be used as features. By default, automatically inferred based on the archive.

`y_col` (`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:*

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

*Arguments:*

`xdt` (`data.table::data.table()`)
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:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## 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))

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

  instance$eval_batch(xdt)

  learner = lrn("regr.km",
    covtype = "matern3_2",
    optim.method = "gen",
    nugget.stability = 10^-8,
    control = list(trace = FALSE))

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

  surrogate$update()

  surrogate$model$model
}
```

---

SurrogateLearnerCollection *Surrogate Model Containing Multiple Learners*

---

## Description

Surrogate model containing multiple mlr3::LearnerRegr. The mlr3::LearnerRegr are fit on the target variables as indicated via `y_cols`. 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 infill 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`.

## 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,
  x_cols = NULL,
  y_cols = NULL
)
```

*Arguments:*

`learners` (list of mlr3::LearnerRegr).

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

`x_cols` (`character()` | `NULL`)
: Column Id's of variables that should be used as features. By default, automatically inferred based on the archive.

`y_cols` (`character()` | `NULL`)
: Column Id's of variables that should be used as targets. By default, automatically inferred based on the archive.

**Method** `predict()`**:** Predict mean response and standard error. Returns a named list of data.tables. Each contains the mean response and standard error for one `y_col`.

*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"))
  objective = ObjectiveRFun$new(fun = fun, domain = domain, codomain = codomain)

  instance = OptimInstanceMultiCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 5))
  xdt = generate_design_random(instance$search_space, n = 4)$data

  instance$eval_batch(xdt)

  learner1 = lrn("regr.km",
    covtype = "matern3_2",
    optim.method = "gen",
    nugget.stability = 10^-8,
    control = list(trace = FALSE))

  learner2 = lrn("regr.ranger",
    num.trees = 500,
    keep.inbag = TRUE,
    se.method = "jack")

  surrogate = srlrnc(list(learner1, learner2), archive = instance$archive)

  surrogate$update()

  surrogate$model

  surrogate$model[["y2"]]$model
}
```

# Index

∗ **Acquisition Function**
    AcqFunction, 5
    mlr_acqfunctions, 14
    mlr_acqfunctions_cb, 14
    mlr_acqfunctions_ei, 16
    mlr_acqfunctions_eips, 18
    mlr_acqfunctions_mean, 20
    mlr_acqfunctions_pi, 22
    mlr_acqfunctions_smsego, 24
∗ **Dictionary**
    mlr_acqfunctions, 14
    mlr_loop_functions, 26
∗ **Loop Function**
    loop_function, 13
    mlr_loop_functions, 26
    mlr_loop_functions_ego, 27
    mlr_loop_functions_mpcl, 30
    mlr_loop_functions_parego, 32
    mlr_loop_functions_smsego, 35
∗ **datasets**
    mlr_acqfunctions, 14
    mlr_loop_functions, 26
∗ **mbo_defaults**
    default_acqfun, 10
    default_acqopt, 10
    default_loopfun, 11
    default_surrogate, 11
    mbo_defaults, 13

acqf, 4
acqf(), *14*, *16*, *18*, *20*, *22*
AcqFunction, *4*, 5, *7–10*, *14–23*, *25*, *28*, *31*, *33*, *37–39*, *42*, *43*
AcqFunctionCB *(*mlr_acqfunctions_cb*)*, 14
AcqFunctionEI *(*mlr_acqfunctions_ei*)*, 16
AcqFunctionEIPS *(*mlr_acqfunctions_eips*)*, 18
AcqFunctionMean *(*mlr_acqfunctions_mean*)*, 20
AcqFunctionPI *(*mlr_acqfunctions_pi*)*, 22
AcqFunctionSmsEgo *(*mlr_acqfunctions_smsego*)*, 24
acqo, 7
AcqOptimizer, *7*, 8, *10*, *28*, *31*, *33*, *36–39*, *42*, *43*

bayesopt_ego, *11*, *13*, *37*
bayesopt_ego *(*mlr_loop_functions_ego*)*, 27
bayesopt_mpcl *(*mlr_loop_functions_mpcl*)*, 30
bayesopt_parego *(*mlr_loop_functions_parego*)*, 32
bayesopt_smsego, *11*
bayesopt_smsego *(*mlr_loop_functions_smsego*)*, 35
bbotk::Archive, *5*, *8*, *28*, *30*, *31*, *33*, *34*, *36*, *38*, *39*, *46–48*, *50*, *53*
bbotk::Objective, *5*, *15*, *16*, *18*, *21*, *22*, *24*
bbotk::OptimInstance, *5*, *6*, *8*, *10–12*, *25*, *38–40*, *42*, *43*, *45–48*, *50*, *53*
bbotk::OptimInstanceMultiCrit, *12*, *33–36*, *45*
bbotk::OptimInstanceSingleCrit, *12*, *18*, *28*, *30*, *31*, *45*
bbotk::Optimizer, *7–9*, *38*
bbotk::OptimizerRandomSearch, *10*
bbotk::Terminator, *7–9*, *36*, *38*
bbotk::TerminatorEvals, *10*, *25*, *36*
bbotk_reflections$optimizer_properties, *38*, *42*

data.table::data.table(), *6*, *9*, *49*, *51*, *53*, *54*
default_acqfun, *10*, 10, *11–13*
default_acqopt, *10*, 10, *11–13*
default_loopfun, *10*, 11, *12*, *13*
default_surrogate, *10*, *11*, 11, *13*
dictionary, *4*, *14*, *16*, *18*, *20*, *22*

loop_function, *11*, 13, *27, 28*, *31*, *34*, *36–39*, *42, 43*

mbo_defaults, *10–12*, 13
mlr3::Learner, *12*
mlr3::LearnerRegr, *46*, *49, 50*, *52, 53*
mlr3::MeasureRegr, *49*, *52*
mlr3::mlr_measures_regr.rsq, *49*, *52*
mlr3mbo *(*mlr3mbo-package*)*, 3
mlr3mbo-package, 3
mlr3mbo::AcqFunction, *15, 16*, *18*, *21, 22*, *24*
mlr3mbo::Surrogate, *50*, *52*
mlr3misc::Dictionary, *14*, *26, 27*
mlr3misc::dictionary_sugar_get(), *4*
mlr3tuning::Tuner, *41*
mlr3tuning::TunerFromOptimizer, *41*
mlr3tuning::TuningInstanceMultiCrit, *38*, *40*, *42, 43*
mlr3tuning::TuningInstanceSingleCrit, *38*, *40*, *42, 43*
mlr_acqfunctions, *4*, *7*, *14*, 14, *15–23*, *25*, *27*
mlr_acqfunctions_cb, *7*, *14*, 14, *17*, *19*, *21*, *23*, *25*
mlr_acqfunctions_ei, *7*, *10*, *14, 15*, 16, *19*, *21*, *23*, *25*, *37*
mlr_acqfunctions_eips, *7*, *14, 15*, *17*, 18, *21*, *23*, *25*
mlr_acqfunctions_mean, *7*, *14, 15*, *17*, *19*, 20, *23*, *25*
mlr_acqfunctions_pi, *7*, *14, 15*, *17*, *19*, *21*, 22, *25*
mlr_acqfunctions_smsego, *7*, *10*, *14, 15*, *17*, *19*, *21*, *23*, 24, *35, 36*
mlr_loop_functions, *13, 14*, 26, *28*, *31*, *34*, *36*
mlr_loop_functions_ego, *13*, *27*, 27, *31*, *34*, *36*
mlr_loop_functions_mpcl, *13*, *27, 28*, 30, *34*, *36*
mlr_loop_functions_parego, *13*, *27, 28*, *31*, 32, *36*
mlr_loop_functions_smsego, *13*, *27, 28*, *31*, *34*, 35
mlr_optimizers_mbo, 37
mlr_reflections$learner_properties, *48*, *50*, *53*
mlr_reflections$task_feature_types, *48*, *50*, *53*
mlr_tuners_mbo, 41

OptimizerMbo, *13*, *27*, *30*, *32*, *35*, *38*, *40–43*, *45*
OptimizerMbo *(*mlr_optimizers_mbo*)*, 37

paradox::Param, *38*, *42*
paradox::ParamSet, *4*, *6–8*, *46–48*

R6, *5*, *9*, *15*, *17*, *19*, *21*, *23*, *25*, *39*, *42*, *48*, *50*, *53*
R6::R6Class, *14*, *27*
requireNamespace(), *39*, *42*, *48*, *50*, *53*
result_by_default, 44
result_by_surrogate_design, *38*, *40*, *42, 43*, 45

srlrn, 45
srlrnc, 46
Surrogate, *5, 6*, *11, 12*, *28*, *31*, *37–39*, *42, 43*, 47
SurrogateLearner, *12*, *15*, *17*, *21*, *23*, *28*, *31*, *33*, *45, 46*, 49
SurrogateLearnerCollection, *12*, *18, 19*, *25*, *36*, *46, 47*, 52

TunerMbo *(*mlr_tuners_mbo*)*, 41