Package 'mlr3mbo'

May 9, 2026

Type Package

Title Flexible Bayesian Optimization

Version 1.1.1

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 mlr3 (>= 1.2.0), mlr3tuning (>= 1.4.0), R (>= 3.1.0)

Imports bbotk (>= 1.8.0), checkmate (>= 2.0.0), data.table, lgr (>= 0.3.4), mlr3misc (>= 0.21.0), paradox (>= 1.0.1), spacefillr, R6 (>= 2.4.1)

Suggests DiceKriging, emoa, fastGHQuad, lhs, mlr3learners (>= 0.12.0), mirai, mlr3pipelines (>= 0.5.2), nloptr, processx, ranger, rgenoud, rpart, redux, rush (>= 1.0.0), stringi, testthat (>= 3.0.0)

ByteCompile no

Encoding UTF-8

Config/testthat/edition 3

Config/testthat/parallel false

NeedsCompilation yes

RoxygenNote 7.3.3

Collate 'mlr_acqfunctions.R' 'AcqFunction.R' 'AcqFunctionAEI.R'
'AcqFunctionCB.R' 'AcqFunctionEHVI.R' 'AcqFunctionEHVIGH.R'
'AcqFunctionEI.R' 'AcqFunctionEILog.R' 'AcqFunctionEIPS.R'
'AcqFunctionMean.R' 'AcqFunctionMulti.R' 'AcqFunctionPI.R'
'AcqFunctionSD.R' 'AcqFunctionSmsEgo.R'
'AcqFunctionStochasticCB.R' 'AcqFunctionStochasticEI.R'
'AcqOptimizer.R' 'mlr_acqoptimizers.R' 'AcqOptimizerDirect.R'
'AcqOptimizerLbfgsb.R' 'AcqOptimizerLocalSearch.R'
'AcqOptimzerRandomSearch.R' 'mlr_input_trafos.R' 'InputTrafo.R'
'InputTrafoUnitcube.R' 'aaa.R' 'OptimizerADBO.R'
'OptimizerMbo.R' 'OptimizerAsyncMbo.R' 'mlr_output_trafos.R'
'OutputTrafo.R' 'OutputTrafoLog.R' 'OutputTrafoStandardize.R'
'mlr_result_assigners.R' 'ResultAssigner.R'
'ResultAssignerArchive.R' 'ResultAssignerSurrogate.R'
'Surrogate.R' 'SurrogateLearner.R'
'SurrogateLearnerCollection.R' 'TunerADBO.R' 'TunerAsyncMbo.R'
'TunerMbo.R' 'mlr_loop_functions.R' 'bayesopt_ego.R'
'bayesopt_emo.R' 'bayesopt_mpcl.R' 'bayesopt_parego.R'
'bayesopt_smsego.R' 'bibentries.R' 'conditions.R' 'helper.R'
'loop_function.R' 'mbo_defaults.R' 'sugar.R' 'zzz.R'

Author Marc Becker [cre, aut] (ORCID: <https://orcid.org/0000-0002-8115-0400>),
Lennart Schneider [aut] (ORCID:
<https://orcid.org/0000-0003-4152-5308>),
Jakob Richter [aut] (ORCID: <https://orcid.org/0000-0003-4481-5554>),
Michel Lang [aut] (ORCID: <https://orcid.org/0000-0001-9754-0393>),
Bernd Bischl [aut] (ORCID: <https://orcid.org/0000-0001-6002-6980>),
Florian Pfisterer [aut] (ORCID:
<https://orcid.org/0000-0001-8867-762X>),
Martin Binder [aut],
Sebastian Fischer [aut] (ORCID:
<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 Marc Becker <marcbecker@posteo.de>

Repository CRAN

Date/Publication 2026-04-24 16:30:09 UTC

Contents

`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: Marc Becker <marcbecker@posteo.de> (ORCID)

Authors:

- Lennart Schneider <lennart.sch@web.de> (ORCID)
- Jakob Richter <jakob1richter@gmail.com> (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.

Value

AcqFunction

Examples

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

acqfs *Syntactic Sugar Acquisition Functions Construction*

Description

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

Usage

```
acqfs(.keys, ...)
```

Arguments

`.keys`	(`character()`) Keys passed to the respective dictionary to retrieve multiple objects.
`...`	(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

List of AcqFunctions

Examples

```
acqfs(c("ei", "pi", "cb"))
```

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 related to the passed bbotk::Archive. 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$reset()`
- `AcqFunction$eval_many()`
- `AcqFunction$eval_dt()`
- `AcqFunction$assert_surrogate()`
- `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 = NULL,
  requires_predict_type_se,
  surrogate_class,
  direction,
  packages = NULL,
```

```
  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.

`requires_predict_type_se` (`logical(1)`)
Whether the acquisition function requires the surrogate to have `"se"` as `$predict_type`.

`surrogate_class` (`character(1)`)
Allowed class of the surrogate.

`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:

```
AcqFunction$update()
```

Method `reset()`**:** Reset the acquisition function.

Can be implemented by subclasses.

Usage:

```
AcqFunction$reset()
```

Method `eval_many()`**:** Evaluates multiple input values on the acquisition 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 acquisition functions and multiple y-columns for multi-objective acquisition 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 acquisition functions and multiple y-columns for multi-objective acquisition functions, e.g. `data.table(y = 1:2)` or `data.table(y1 = 1:2, y2 = 3:4)`.

Method `assert_surrogate()`**:** Validate that the surrogate is compatible with this acquisition function. Asserts the surrogate class and that `$predict_type` is `"se"` if required. Subclasses with additional requirements must override this method.

Usage:

```
AcqFunction$assert_surrogate(surrogate)
```

Arguments:

`surrogate` (Surrogate)
Surrogate to validate.

Returns: The validated Surrogate.

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_ei_log`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_multi`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_sd`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions_stochastic_cb`, `mlr_acqfunctions_stochast`

---

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, callbacks = NULL, ...)
```

Arguments

optimizer	(bbotk::OptimizerBatch | character(1)) bbotk::OptimizerBatch that is to be used or the name of the optimizer in mlr_acqoptimizers. If optimizer is missing, the mlr_acqoptimizers dictionary is returned.
terminator	(bbotk::Terminator) bbotk::Terminator that is to be used.
acq_function	(NULL | AcqFunction) AcqFunction that is to be used. Can also be NULL.
callbacks	(NULL | list of mlr3misc::Callback) Callbacks used during acquisition function optimization.
...	(named list()) 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 acquisition function optimization. Wraps an bbotk::OptimizerBatch 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::ArchiveBatch of the acquisition function bbotk::OptimInstanceBatch. Note that setting `n_candidates > 1` is usually not a sensible idea but it is still supported for experimental reasons. Note that in the case of the acquisition function bbotk::OptimInstanceBatch being multi-objective, due to using an AcqFunctionMulti, selection of the best candidates is performed via non-dominated-sorting. 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 (which is contained in the archive of the AcqFunction)? This is sensible when using a population based acquisition function optimizer, e.g., local search or mutation. Default is `FALSE`. Note that in the case of the bbotk::OptimInstance being multi-objective, selection of the best point(s) is performed via non-dominated-sorting.

`warmstart_size integer(1) | "all"`
: Number of best points selected from the bbotk::Archive of the actual bbotk::OptimInstance that are to be used for warm starting. Can either be an integer or "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(s) resulting of the acquisition function optimization were already evaluated on the actual bbotk::OptimInstance. Should such candidate proposals be ignored and only candidates that were yet not evaluated be considered? 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, e.g., by proposing a randomly sampled point for evaluation? Setting this to `FALSE` can be helpful for debugging. Default is `TRUE`.

Public fields

`optimizer` (bbotk::OptimizerBatch).

`terminator` (bbotk::Terminator).

`acq_function` (AcqFunction).

`callbacks` (`NULL` | list of mlr3misc::Callback).

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$reset()`
- `AcqOptimizer$clone()`

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

Usage:

```
AcqOptimizer$new(optimizer, terminator, acq_function = NULL, callbacks = NULL)
```

Arguments:

`optimizer` (bbotk::OptimizerBatch).

`terminator` (bbotk::Terminator).

`acq_function` (`NULL` | AcqFunction).

`callbacks` (`NULL` | list of mlr3misc::Callback)

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

Usage:

```
AcqOptimizer$format()
```

Returns: (`character(1)`).

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 candidate.

Method `reset()`**:** Reset the acquisition function optimizer.
Currently not used.

Usage:

```
AcqOptimizer$reset()
```

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.

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 = OptimInstanceBatchSingleCrit$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(
  optimizer = opt("random_search", batch_size = 1000),
  terminator = trm("evals", n_evals = 1000),
  acq_function = acq_function)

acq_optimizer$optimize()
}
```

`AcqOptimizerDirect` *Direct Acquisition Function Optimizer*

Description

Direct acquisition function optimizer. Calls `nloptr()` from **nloptr**. In its default setting, the algorithm restarts `5 * D` times and runs at most for `100 * D^2` function evaluations, where `D` is the dimension of the search space. Each run stops when the relative tolerance of the parameters is less than `10^-4`. The first iteration starts with the best point in the archive and the next iterations start from a random point.

Parameters

`restart_strategy character(1)`
: Restart strategy. Can be `"none"` or `"random"`. Default is `"none"`.

`max_restarts integer(1)`
: Maximum number of restarts. Default is `5 * D` (Default).

Termination Parameters

The following termination parameters can be used.

`stopval numeric(1)`
: Stop value. Deactivate with `-Inf` (Default).

`maxeval integer(1)`
: Maximum number of evaluations. Default is `100 * D^2`, where `D` is the dimension of the search space. Deactivate with `-1L`.

`xtol_rel numeric(1)`
: Relative tolerance of the parameters. Default is `10^-4`. Deactivate with `-1`.

`xtol_abs numeric(1)`
: Absolute tolerance of the parameters. Deactivate with `-1` (Default).

`ftol_rel numeric(1)`
: Relative tolerance of the objective function. Deactivate with `-1`. (Default).

`ftol_abs numeric(1)`
: Absolute tolerance of the objective function. Deactivate with `-1` (Default).

Super class

`mlr3mbo::AcqOptimizer` -> `AcqOptimizerDirect`

Public fields

`state (list())`
: List of `nloptr::nloptr()` results.

Active bindings

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

Methods

Public methods:

- `AcqOptimizerDirect$new()`
- `AcqOptimizerDirect$optimize()`
- `AcqOptimizerDirect$clone()`

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

Usage:

```
AcqOptimizerDirect$new(acq_function = NULL)
```

Arguments:

`acq_function` (`NULL` | AcqFunction).

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

Usage:

`AcqOptimizerDirect$optimize()`

Returns: `data.table::data.table()` with 1 row per candidate.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

If the restart strategy is `"none"`, the optimizer starts with the best point in the archive. The optimization stops when one of the stopping criteria is met.

If `restart_strategy` is `"random"`, the optimizer runs at least for `maxeval` iterations. The first iteration starts with the best point in the archive and stops when one of the stopping criteria is met. The next iterations start from a random point.

Examples

```
if (requireNamespace("nloptr")) {
  acqo("direct")
}
```

`AcqOptimizerLbfgsb` *L-BFGS-B Acquisition Function Optimizer*

Description

L-BFGS-B acquisition function optimizer. Calls `nloptr()` from **nloptr**. In its default setting, the algorithm restarts `5 * D` times and runs at most for `100 * D^2` function evaluations, where `D` is the dimension of the search space. Each run stops when the relative tolerance of the parameters is less than `10^-4`. The first iteration starts with the best point in the archive and the next iterations start from a random point.

Parameters

`restart_strategy character(1)`
: Restart strategy. Can be `"none"` or `"random"`. Default is `"none"`.

`max_restarts integer(1)`
: Maximum number of restarts. Default is `5 * D` (Default).

Termination Parameters

The following termination parameters can be used.

`stopval numeric(1)`
: Stop value. Deactivate with `-Inf` (Default).

`maxeval integer(1)`
: Maximum number of evaluations. Default is `100 * D^2`, where `D` is the dimension of the search space. Deactivate with `-1L`.

`xtol_rel numeric(1)`
: Relative tolerance of the parameters. Default is `10^-4`. Deactivate with `-1`.

`xtol_abs numeric(1)`
: Absolute tolerance of the parameters. Deactivate with `-1` (Default).

`ftol_rel numeric(1)`
: Relative tolerance of the objective function. Deactivate with `-1`. (Default).

`ftol_abs numeric(1)`
: Absolute tolerance of the objective function. Deactivate with `-1` (Default).

Super class

`mlr3mbo::AcqOptimizer` -> `AcqOptimizerLbfgsb`

Public fields

`state (list())`
: List of `nloptr::nloptr()` results.

Active bindings

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

Methods

Public methods:

- `AcqOptimizerLbfgsb$new()`
- `AcqOptimizerLbfgsb$optimize()`
- `AcqOptimizerLbfgsb$clone()`

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

Usage:

```
AcqOptimizerLbfgsb$new(acq_function = NULL)
```

Arguments:

`acq_function` (`NULL` | AcqFunction).

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

Usage:

```
AcqOptimizerLbfgsb$optimize()
```

Returns: `data.table::data.table()` with 1 row per candidate.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

If the restart strategy is `"none"`, the optimizer starts with the best point in the archive. The optimization stops when one of the stopping criteria is met.

If `restart_strategy` is `"random"`, the optimizer runs at least for `maxeval` iterations. The first iteration starts with the best point in the archive and stops when one of the stopping criteria is met. The next iterations start from a random point.

Examples

```
if (requireNamespace("nloptr")) {
  acqo("lbfgsb")
}
```

AcqOptimizerLocalSearch

Local Search Acquisition Function Optimizer

Description

Local search acquisition function optimizer. Calls `bbotk::local_search()`. For the meaning of the control parameters, see `bbotk::local_search_control()`. The termination stops when the budget defined by the `n_searches`, `n_steps`, and `n_neighs` parameters is exhausted.

Super class

`mlr3mbo::AcqOptimizer` -> `AcqOptimizerLocalSearch`

Public fields

`state` (`list()`)
: List of `cmaes::cma_es()` results.

Active bindings

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

Methods

Public methods:

- `AcqOptimizerLocalSearch$new()`
- `AcqOptimizerLocalSearch$optimize()`
- `AcqOptimizerLocalSearch$reset()`
- `AcqOptimizerLocalSearch$clone()`

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

Usage:

```
AcqOptimizerLocalSearch$new(acq_function = NULL)
```

Arguments:

`acq_function` (`NULL` | AcqFunction).

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

Usage:

```
AcqOptimizerLocalSearch$optimize()
```

Returns: `data.table::data.table()` with 1 row per candidate.

Method `reset()`**:** Reset the acquisition function optimizer.
Currently not used.

Usage:

```
AcqOptimizerLocalSearch$reset()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
acqo("local_search")
```

`AcqOptimizerRandomSearch`

Random Search Acquisition Function Optimizer

Description

Random search acquisition function optimizer. By default, it samples `100 * D^2` random points in the search space, where `D` is the dimension of the search space. The point with the highest acquisition value is returned.

Parameters

`n_evals integer(1)`
: Number of random points to sample. Default is `100 * D^2`, where `D` is the dimension of the search space.

Super class

`mlr3mbo::AcqOptimizer` -> `AcqOptimizerRandomSearch`

Active bindings

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

Methods

Public methods:

- `AcqOptimizerRandomSearch$new()`
- `AcqOptimizerRandomSearch$optimize()`
- `AcqOptimizerRandomSearch$clone()`

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

Usage:

```
AcqOptimizerRandomSearch$new(acq_function = NULL)
```

Arguments:

`acq_function` (`NULL` | AcqFunction).

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

Usage:

```
AcqOptimizerRandomSearch$optimize()
```

Returns: `data.table::data.table()` with 1 row per candidate.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
acqo("random_search")
```

default_acqfunction *Default Acquisition Function*

Description

Chooses a default acquisition function, i.e. the criterion used to propose future points. For synchronous single-objective optimization and a purely numeric parameter space, defaults to mlr_acqfunctions_cb with `lambda = 3`. For synchronous single-objective optimization and a mixed numeric-categorical parameter space, defaults to mlr_acqfunctions_cb with `lambda = 1`. For synchronous multi-objective optimization, defaults to mlr_acqfunctions_smsego. For asynchronous single-objective optimization, defaults to mlr_acqfunctions_stochastic_cb.

Usage

```
default_acqfunction(instance)
```

Arguments

`instance`	(bbotk::OptimInstance). An object that inherits from 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 AcqOptimizerLocalSearch with `n_searches = 10`, `n_steps = ceiling(100 * D^2 / 300)`, and `n_neighs = 30`, where `D` is the dimension of the search space.

Usage

```
default_acqoptimizer(acq_function, instance)
```

Arguments

`acq_function`	(AcqFunction).
`instance`	(bbotk::OptimInstance).

Value

AcqOptimizer

See Also

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

---

default_gp *Default Gaussian Process*

Description

This is a helper function that constructs a default Gaussian Process mlr3learners::LearnerRegrKM 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

mlr3learners::LearnerRegrKM

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

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 mlr3learners::LearnerRegrRanger which is for example used in default_surrogate().

Usage

```
default_rf(noisy = FALSE)
```

Arguments

noisy	(logical(1)) Whether the learner will be used in a noisy objective function scenario. See default_surrogate().

Value

mlr3learners::LearnerRegrRanger

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*

Description

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

For purely numeric (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 the internal NA handling method `na.action = "na_learn"` of **ranger**.

If `n_learner` is 1, the learner is wrapped as a SurrogateLearner. Otherwise, if `n_learner` is larger than 1, multiple deep clones of the learner are wrapped as a SurrogateLearnerCollection.

Usage

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

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 Surrogate. If not specified will be based on the number of objectives as stated by the instance.
`force_random_forest`	(`logical(1)`). If `TRUE`, a random forest is constructed even if the parameter space is purely numeric.

Value

Surrogate

See Also

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

InputTrafo *Input Transformation Base Class*

Description

Abstract input transformation class.

An input transformation can be used within a Surrogate to perform a transformation of the feature variables.

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()`.

`state` (named `list()` | `NULL`)
: List of meta information regarding the parameters and their state.

`search_space` (paradox::ParamSet)
: Search space.

`cols_x` (paradox::ParamSet)
: Column ids of feature variables that should be transformed.

Methods

Public methods:

- `InputTrafo$new()`
- `InputTrafo$update()`
- `InputTrafo$transform()`
- `InputTrafo$format()`
- `InputTrafo$print()`
- `InputTrafo$clone()`

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

Usage:

```
InputTrafo$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 `update()`**:** Learn the transformation based on observed data and update parameters in `$state`. Must be implemented by subclasses.

Usage:

```
InputTrafo$update(xdt)
```

Arguments:

`xdt` (`data.table::data.table()`)
: Data. One row per observation with at least columns `$cols_x`.

Method `transform()`**:** Perform the transformation. Must be implemented by subclasses.

Usage:

```
InputTrafo$transform(xdt)
```

Arguments:

`xdt` (`data.table::data.table()`)
Data. One row per observation with at least columns `$cols_x`.

Returns: `data.table::data.table()` with the transformation applied to the columns `$cols_x` (if applicable) or a subset thereof.

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

Usage:

```
InputTrafo$format()
```

Returns: (`character(1)`).

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

Usage:

```
InputTrafo$print()
```

Returns: (`character()`).

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Input Transformation: `InputTrafoUnitcube`, `mlr_input_trafos`

`InputTrafoUnitcube` *Input Transformation Unitcube*

Description

Input transformation that performs for each numeric and integer feature min-max scaling to `[\0, 1\]` based on the boundaries of the search space.

[\0, 1]: R:%5C%5C0,%201%5C

Super class

`mlr3mbo::InputTrafo` -> `InputTrafoUnitcube`

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 `requireNamespace()`.

Methods

Public methods:

- `InputTrafoUnitcube$new()`
- `InputTrafoUnitcube$update()`
- `InputTrafoUnitcube$transform()`
- `InputTrafoUnitcube$clone()`

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

Usage:

```
InputTrafoUnitcube$new()
```

Method `update()`**:** Learn the transformation based on observed data and update parameters in `$state`.

Usage:

```
InputTrafoUnitcube$update(xdt)
```

Arguments:

`xdt` (`data.table::data.table()`)
: Data. One row per observation with at least columns `$cols_x`.

Method `transform()`**:** Perform the transformation.

Usage:

```
InputTrafoUnitcube$transform(xdt)
```

Arguments:

`xdt` (`data.table::data.table()`)
: Data. One row per observation with at least columns `$cols_x`.

Returns: `data.table::data.table()` with the transformation applied to the columns `$cols_x` (if applicable) or a subset thereof.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Input Transformation: `InputTrafo`, `mlr_input_trafos`

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 = OptimInstanceBatchSingleCrit$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()

  input_trafo = it("unitcube")

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

  surrogate$update()

  surrogate$input_trafo$state

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

`it` *Syntactic Sugar Input Trafo Construction*

Description

This function complements mlr_input_trafos with functions in the spirit of `mlr_sugar` from **mlr3**.

Usage

```
it(.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

InputTrafo

Examples

```
it("unitcube")
```

---

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()`

---

mlr3mbo_conditions *Condition Classes for mlr3mbo*

Description

Condition classes for mlr3mbo.

Usage

```
error_random_interleave(msg, ..., class = NULL, signal = TRUE, parent = NULL)

error_surrogate_update(msg, ..., class = NULL, signal = TRUE, parent = NULL)

error_acq_optimizer(msg, ..., class = NULL, signal = TRUE, parent = NULL)
```

Arguments

`msg`	(`character(1)`) Error message.
`...`	(any) Passed to `sprintf()`.
`class`	(`character`) Additional class(es).
`signal`	(`logical(1)`) If `FALSE`, the condition object is returned instead of being signaled.
`parent`	(`condition`) Parent condition.

Errors

- `error_random_interleave()` for the `Mlr3ErrorMboRandomInterleave` class, signalling a random interleave error.
- `error_surrogate_update()` for the `Mlr3ErrorMboSurrogateUpdate` class, signalling a surrogate update error.
- `error_acq_optimizer()` for the `Mlr3ErrorMboAcqOptimizer` class, signalling an acquisition function optimizer error.

`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()` and `acqfs()`.

Format

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

Methods

See mlr3misc::Dictionary.

See Also

Sugar functions: `acqf()`, `acqfs()`

Other Dictionary: `mlr_acqoptimizers`, `mlr_input_trafos`, `mlr_loop_functions`, `mlr_output_trafos`, `mlr_result_assigners`

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions_aei`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_ehvi`, `mlr_acqfunctions_ehvigh`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_ei_log`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_multi`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_sd`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions_stochastic_cb`, `mlr_acqfunctions_stochastic_ei`

Examples

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

`mlr_acqfunctions_aei` *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(): Update the acquisition function and set 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:

deep Whether to make a deep clone.

References

- Huang D, Allen TT, Notz WI, Zheng N (2012). "Erratum To: Global Optimization of Stochastic Black-box Systems via Sequential Kriging Meta-Models." *Journal of Global Optimization*, **54**(2), 431–431.

See Also

Other Acquisition Function: AcqFunction, mlr_acqfunctions, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_ei_log, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_multi, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego, mlr_acqfunctions_stochastic_cb, mlr_acqfunctions_stochastic_ei

Examples

```
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 = OptimInstanceBatchSingleCrit$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)))
}
```

`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)`)
  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:

```
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, mlr_acqfunctions_aei, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_ei_log, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_multi, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego, mlr_acqfunctions_stochastic_cb, mlr_acqfunctions_stochastic_ei

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 = OptimInstanceBatchSingleCrit$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)))
}
```

`mlr_acqfunctions_ehvi` *Acquisition Function Expected Hypervolume Improvement*

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()`**:** Update the acquisition function and set `ys_front` and `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

- Emmerich, Michael, Yang, Kaifeng, Deutz, André, Wang, Hao, Fonseca, M. C (2016). "A Multicriteria Generalization of Bayesian Global Optimization." In Pardalos, M. P, Zhigljavsky, Anatoly, Žilinskas, Julius (eds.), *Advances in Stochastic and Deterministic Global Optimization*, 229–242. Springer International Publishing, Cham.

See Also

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions`, `mlr_acqfunctions_aei`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_ehvigh`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_ei_log`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_multi`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_sd`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions_stochastic_cb`, `mlr_acqfunctions_stochastic_ei`

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 = OptimInstanceBatchMultiCrit$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

Expected Hypervolume Improvement. Computed via Gauss-Hermite quadrature.

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()`**:** Update the acquisition function and set `ys_front`, `ref_point`, `hypervolume` and `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

- Rahat, Alma, Chugh, Tinkle, Fieldsend, Jonathan, Allmendinger, Richard, Miettinen, Kaisa (2022). "Efficient Approximation of Expected Hypervolume Improvement using Gauss-Hermit Quadrature." In Rudolph, Günter, Kononova, V. A, Aguirre, Hernán, Kerschke, Pascal, Ochoa, Gabriela, Tušar, Tea (eds.), *Parallel Problem Solving from Nature – PPSN XVII*, 90–103.

See Also

Other Acquisition Function: `AcqFunction`, `mlr_acqfunctions`, `mlr_acqfunctions_aei`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_ehvi`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_ei_log`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_multi`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_sd`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions_stochastic_cb`, `mlr_acqfunctions_stochastic_ei`

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 = OptimInstanceBatchMultiCrit$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")
```

Parameters

- "epsilon" (numeric(1))
  ϵ value used to determine the amount of exploration. Higher values result in the importance of improvements predicted by the posterior mean decreasing relative to the importance of potential improvements in regions of high predictive uncertainty. Defaults to 0 (standard Expected Improvement).

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, epsilon = 0)
```

Arguments:

surrogate (NULL | SurrogateLearner).

epsilon (numeric(1)).

Method update(): Update the acquisition function and set 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, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei_log, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_multi, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego, mlr_acqfunctions_stochastic_cb, mlr_acqfunctions_stochastic_ei

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 = OptimInstanceBatchSingleCrit$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::OptimInstanceBatchSingleCrit. Additionally to target values of the codomain that should be minimized or maximized, the bbotk::Objective of the bbotk::OptimInstanceBatchSingleCrit 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()`:

```
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()`**:** Update the acquisition function and set `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, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_ei_log, mlr_acqfunctions_mean, mlr_acqfunctions_multi, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego, mlr_acqfunctions_stochastic_cb, mlr_acqfunctions_stochastic_ei

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 = OptimInstanceBatchSingleCrit$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_ei_log`
Acquisition Function Expected Improvement on Log Scale

Description

Expected Improvement assuming that the target variable has been modeled on log scale. In general only sensible if the SurrogateLearner uses an OutputTrafoLog without inverting the posterior predictive distribution (`invert_posterior = FALSE`). See also the example below.

Dictionary

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

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

Parameters

- `"epsilon"` (`numeric(1)`)
  ϵ value used to determine the amount of exploration. Higher values result in the importance of improvements predicted by the posterior mean decreasing relative to the importance of potential improvements in regions of high predictive uncertainty. Defaults to `0` (standard Expected Improvement).

Super classes

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

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:

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

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

Usage:

```
AcqFunctionEILog$new(surrogate = NULL, epsilon = 0)
```

Arguments:

surrogate (NULL | SurrogateLearner).

epsilon (numeric(1)).

Method update(): Update the acquisition function and set y_best.

Usage:

```
AcqFunctionEILog$update()
```

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

Usage:

```
AcqFunctionEILog$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_multi, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego, mlr_acqfunctions_stochastic_cb, mlr_acqfunctions_stochastic_ei

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

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

  learner = default_gp()

  output_trafo = ot("log", invert_posterior = FALSE)

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

```
  acq_function = acqf("ei_log", 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_ei_log, mlr_acqfunctions_eips, mlr_acqfunctions_multi, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego, mlr_acqfunctions_stochastic_cb, mlr_acqfunctions_stochastic_ei

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 = OptimInstanceBatchSingleCrit$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_multi

Acquisition Function Wrapping Multiple Acquisition Functions

Description

Wrapping multiple AcqFunctions resulting in a multi-objective acquisition function composed of the individual ones. Note that the optimization direction of each wrapped acquisition function is corrected for maximization.

For each acquisition function, the same Surrogate must be used. If acquisition functions passed during construction already have been initialized with a surrogate, it is checked whether the surrogate is the same for all acquisition functions. If acquisition functions have not been initialized with a surrogate, the surrogate passed during construction or lazy initialization will be used for all acquisition functions.

For optimization, AcqOptimizer can be used as for any other AcqFunction, however, the bbotk::OptimizerBatch wrapped within the AcqOptimizer must support multi-objective optimization as indicated via the `multi-crit` property.

Dictionary

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

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

Super classes

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

Active bindings

`surrogate` (Surrogate)
: Surrogate.

`acq_functions` (list of AcqFunction)
: Points to the list of the individual acquisition functions.

`acq_function_ids` (character())
: Points to the ids of the individual acquisition functions.

Methods

Public methods:

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

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

Usage:

```
AcqFunctionMulti$new(acq_functions, surrogate = NULL)
```

Arguments:

`acq_functions` (list of AcqFunctions).

`surrogate` (`NULL` | Surrogate).

Method `update()`**:** Update each of the wrapped acquisition functions.

Usage:

```
AcqFunctionMulti$update()
```

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

Usage:

```
AcqFunctionMulti$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_ei_log`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_sd`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions_stochastic_cb`, `mlr_acqfunctions_stochastic_ei`

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 = OptimInstanceBatchSingleCrit$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("multi",
    acq_functions = acqfs(c("ei", "pi", "cb")),
    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. 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:

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

Arguments:

`surrogate` (`NULL` | SurrogateLearner).

Method `update()`**:** Update the acquisition function and set `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, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_ei_log, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_multi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego, mlr_acqfunctions_stochastic_cb, mlr_acqfunctions_stochastic_ei

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 = OptimInstanceBatchSingleCrit$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()`:

```
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:

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

Arguments:

`surrogate` (`NULL` | SurrogateLearner).

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

Usage:

```
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_ei_log`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_multi`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions_stochastic_cb`, `mlr_acqfunctions_stochastic_ei`

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 = OptimInstanceBatchSingleCrit$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("sd", 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)`)
  λ 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)`)
  ϵ 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).

Note

- This acquisition function always also returns its current epsilon values in a list column (`acq_epsilon`). These values will be logged into the bbotk::ArchiveBatch of the bbotk::OptimInstanceBatch of the AcqOptimizer and therefore also in the bbotk::Archive of the actual bbotk::OptimInstance that is to be optimized.

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(1))`
: Optimization progress (typically, the number of function evaluations left). Note that this requires the bbotk::OptimInstanceBatch to be terminated via a bbotk::TerminatorEvals.

Methods

Public methods:

- `AcqFunctionSmsEgo$new()`
- `AcqFunctionSmsEgo$update()`
- `AcqFunctionSmsEgo$reset()`
- `AcqFunctionSmsEgo$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()`**:** Update the acquisition function and set `ys_front`, `ref_point` and `epsilon`.

Usage:

```
AcqFunctionSmsEgo$update()
```

Method `reset()`**:** Reset the acquisition function. Resets `epsilon`.

Usage:

```
AcqFunctionSmsEgo$reset()
```

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`, `mlr_acqfunctions_aei`, `mlr_acqfunctions_cb`, `mlr_acqfunctions_ehvi`, `mlr_acqfunctions_ehvigh`, `mlr_acqfunctions_ei`, `mlr_acqfunctions_ei_log`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_multi`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_sd`, `mlr_acqfunctions_stochastic_cb`, `mlr_acqfunctions_stochastic_ei`

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 = OptimInstanceBatchMultiCrit$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_acqfunctions_stochastic_cb`
Acquisition Function Stochastic Confidence Bound

Description

Lower / Upper Confidence Bound with lambda sampling and decay. The initial λ is drawn from an uniform distribution between `min_lambda` and `max_lambda` or from an exponential distribution with rate `1 / lambda`. λ is updated after each update by the formula `lambda * exp(-rate * (t %% period))`, where `t` is the number of times the acquisition function has been updated.

While this acquisition function usually would be used within an asynchronous optimizer, e.g., OptimizerAsyncMbo, it can in principle also be used in synchronous optimizers, e.g., OptimizerMbo.

Dictionary

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

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

Parameters

- `"lambda"` (`numeric(1)`)
  λ value for sampling from the exponential distribution. Defaults to `1.96`.
- `"min_lambda"` (`numeric(1)`)
  Minimum value of λ for sampling from the uniform distribution. Defaults to `0.01`.
- `"max_lambda"` (`numeric(1)`)
  Maximum value of λ for sampling from the uniform distribution. Defaults to `10`.
- `"distribution"` (`character(1)`)
  Distribution to sample λ from. One of `c("uniform", "exponential")`. Defaults to `uniform`.
- `"rate"` (`numeric(1)`)
  Rate of the exponential decay. Defaults to `0` i.e. no decay.
- `"period"` (`integer(1)`)
  Period of the exponential decay. Defaults to `NULL`, i.e., the decay has no period.

Note

- This acquisition function always also returns its current (`acq_lambda`) and original (`acq_lambda_0`) λ. These values will be logged into the bbotk::ArchiveBatch of the bbotk::OptimInstanceBatch of the AcqOptimizer and therefore also in the bbotk::Archive of the actual bbotk::OptimInstance that is to be optimized.

Super classes

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

Methods

Public methods:

- `AcqFunctionStochasticCB$new()`
- `AcqFunctionStochasticCB$update()`
- `AcqFunctionStochasticCB$reset()`
- `AcqFunctionStochasticCB$clone()`

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

Usage:

```
AcqFunctionStochasticCB$new(
  surrogate = NULL,
  lambda = 1.96,
  min_lambda = 0.01,
  max_lambda = 10,
  distribution = "uniform",
  rate = 0,
  period = NULL
)
```

Arguments:

`surrogate` (`NULL` | SurrogateLearner).

`lambda` (`numeric(1)`).

`min_lambda` (`numeric(1)`).

`max_lambda` (`numeric(1)`).

`distribution` (`character(1)`).

`rate` (`numeric(1)`).

`period` (`NULL` | `integer(1)`).

Method `update()`**:** Update the acquisition function. Samples and decays lambda.

Usage:

```
AcqFunctionStochasticCB$update()
```

Method `reset()`**:** Reset the acquisition function. Resets the private update counter `.t` used within the epsilon decay.

Usage:

```
AcqFunctionStochasticCB$reset()
```

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

Usage:

```
AcqFunctionStochasticCB$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.
- Egelé, Romain, Guyon, Isabelle, Vishwanath, Venkatram, Balaprakash, Prasanna (2023). "Asynchronous Decentralized Bayesian Optimization for Large Scale Hyperparameter Optimization." In *2023 IEEE 19th International Conference on e-Science (e-Science)*, 1–10.

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_ei_log`, `mlr_acqfunctions_eips`, `mlr_acqfunctions_mean`, `mlr_acqfunctions_multi`, `mlr_acqfunctions_pi`, `mlr_acqfunctions_sd`, `mlr_acqfunctions_smsego`, `mlr_acqfunctions_stochastic_ei`

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 = OptimInstanceBatchSingleCrit$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("stochastic_cb", surrogate = surrogate, lambda = 3)

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

`mlr_acqfunctions_stochastic_ei`
Acquisition Function Stochastic Expected Improvement

Description

Expected Improvement with epsilon decay. ϵ is updated after each update by the formula `epsilon * exp(-rate * (t %% period))` where `t` is the number of times the acquisition function has been updated.

While this acquisition function usually would be used within an asynchronous optimizer, e.g., OptimizerAsyncMbo, it can in principle also be used in synchronous optimizers, e.g., OptimizerMbo.

Dictionary

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

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

Parameters

- `"epsilon"` (`numeric(1)`)
  ϵ value used to determine the amount of exploration. Higher values result in the importance of improvements predicted by the posterior mean decreasing relative to the importance of potential improvements in regions of high predictive uncertainty. Defaults to `0.1`.
- `"rate"` (`numeric(1)`)
  Rate of the exponential decay. Defaults to `0.05`.
- `"period"` (`integer(1)`)
  Period of the exponential decay. Defaults to `NULL`, i.e., the decay has no period.

Note

- This acquisition function always also returns its current (`acq_epsilon`) and original (`acq_epsilon_0`) ϵ. These values will be logged into the bbotk::ArchiveBatch of the bbotk::OptimInstanceBatch of the AcqOptimizer and therefore also in the bbotk::Archive of the actual bbotk::OptimInstance that is to be optimized.

Super classes

bbotk::Objective -> mlr3mbo::AcqFunction -> AcqFunctionStochasticEI

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:

- AcqFunctionStochasticEI$new()
- AcqFunctionStochasticEI$update()
- AcqFunctionStochasticEI$reset()
- AcqFunctionStochasticEI$clone()

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

Usage:

```
AcqFunctionStochasticEI$new(
  surrogate = NULL,
  epsilon = 0.1,
  rate = 0.05,
  period = NULL
)
```

Arguments:

surrogate (NULL | SurrogateLearner).

epsilon (numeric(1)).

rate (numeric(1)).

period (NULL | integer(1)).

Method update(): Update the acquisition function. Sets y_best to the best observed objective function value. Decays epsilon.

Usage:

```
AcqFunctionStochasticEI$update()
```

Method reset(): Reset the acquisition function. Resets the private update counter .t used within the epsilon decay.

Usage:

```
AcqFunctionStochasticEI$reset()
```

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

Usage:

```
AcqFunctionStochasticEI$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, mlr_acqfunctions_aei, mlr_acqfunctions_cb, mlr_acqfunctions_ehvi, mlr_acqfunctions_ehvigh, mlr_acqfunctions_ei, mlr_acqfunctions_ei_log, mlr_acqfunctions_eips, mlr_acqfunctions_mean, mlr_acqfunctions_multi, mlr_acqfunctions_pi, mlr_acqfunctions_sd, mlr_acqfunctions_smsego, mlr_acqfunctions_stochastic_cb

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 = OptimInstanceBatchSingleCrit$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("stochastic_ei", surrogate = surrogate)

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

---

mlr_acqoptimizers *Dictionary of Acquisition Function Optimizers*

Description

A simple mlr3misc::Dictionary storing objects of class AcqOptimizer. Each input transformation has an associated help page, see `mlr_acqoptimizers[id]`.

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

Format

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

Methods

See mlr3misc::Dictionary.

See Also

Sugar function: `acqo()`

Other Dictionary: `mlr_acqfunctions`, `mlr_input_trafos`, `mlr_loop_functions`, `mlr_output_trafos`, `mlr_result_assigners`

Examples

```
library(data.table)
as.data.table(mlr_acqoptimizers)
acqo("local_search")
```

---

mlr_input_trafos *Dictionary of Input Transformations*

Description

A simple mlr3misc::Dictionary storing objects of class InputTrafo. Each input transformation has an associated help page, see `mlr_input_trafos[id]`.

For a more convenient way to retrieve and construct an input trafo, see `it()`.

Format

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

Methods

See mlr3misc::Dictionary.

See Also

Sugar function: `it()`

Other Dictionary: `mlr_acqfunctions`, `mlr_acqoptimizers`, `mlr_loop_functions`, `mlr_output_trafos`, `mlr_result_assigners`

Other Input Transformation: `InputTrafo`, `InputTrafoUnitcube`

Examples

```
library(data.table)
as.data.table(mlr_input_trafos)
it("unitcube")
```

`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) 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_acqoptimizers`, `mlr_input_trafos`, `mlr_output_trafos`, `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_smsego`

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::OptimInstanceBatchSingleCrit) The bbotk::OptimInstanceBatchSingleCrit 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::ArchiveBatch 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::ArchiveBatch of the bbotk::OptimInstanceBatchSingleCrit.

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

  surrogate = default_surrogate(instance, n_learner = 2)
  surrogate$cols_y = 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_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::OptimInstanceBatchMultiCrit) The bbotk::OptimInstanceBatchMultiCrit 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::ArchiveBatch 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::ArchiveBatch of the bbotk::OptimInstanceBatchMultiCrit.

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 = OptimInstanceBatchMultiCrit$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)
}
```

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

Arguments

`instance`	(bbotk::OptimInstanceBatchSingleCrit) The bbotk::OptimInstanceBatchSingleCrit 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::ArchiveBatch 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::ArchiveBatch of the bbotk::OptimInstanceBatchSingleCrit.
- To make use of parallel evaluations in the case of 'q > 1, the objective function of the bbotk::OptimInstanceBatchSingleC 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`, `mlr_loop_functions_ego`, `mlr_loop_functions_emo`, `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 = OptimInstanceBatchSingleCrit$new(
    objective = objective,
    terminator = trm("evals", n_evals = 7))

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

Arguments

`instance` (bbotk::OptimInstanceBatchMultiCrit)
The bbotk::OptimInstanceBatchMultiCrit 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::ArchiveBatch 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::ArchiveBatch of the bbotk::OptimInstanceBatchMultiCrit.
- The scalarizations of the objective function values are stored as the `y_scal` column in the bbotk::ArchiveBatch of the bbotk::OptimInstanceBatchMultiCrit.

- To make use of parallel evaluations in the case of 'q > 1, the objective function of the bbotk::OptimInstanceBatchMultiCr 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, mlr_loop_functions_ego, mlr_loop_functions_emo, mlr_loop_functions_mpcl, 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 = OptimInstanceBatchMultiCrit$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", batch_size = 100),
    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,
  surrogate,
  acq_function,
  acq_optimizer,
  init_design_size = NULL,
  random_interleave_iter = 0L
)
```

Arguments

`instance`	(bbotk::OptimInstanceBatchMultiCrit) The bbotk::OptimInstanceBatchMultiCrit 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::ArchiveBatch 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::ArchiveBatch of the bbotk::OptimInstanceBatchMultiCrit.
- Due to the iterative computation of the epsilon within the mlr_acqfunctions_smsego, requires the bbotk::Terminator of the bbotk::OptimInstanceBatchMultiCrit 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`, `mlr_loop_functions_ego`, `mlr_loop_functions_emo`, `mlr_loop_functions_mpcl`, `mlr_loop_functions_parego`

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 = OptimInstanceBatchMultiCrit$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)
}
```

`mlr_optimizers_adbo` *Asynchronous Decentralized Bayesian Optimization*

Description

`OptimizerADBO` class that implements Asynchronous Decentralized Bayesian Optimization (ADBO). ADBO is a variant of Asynchronous Model Based Optimization (AMBO) that uses AcqFunction-StochasticCB with exponential lambda decay.

Currently, only single-objective optimization is supported and OptimizerADBO is considered an experimental feature and API might be subject to changes.

Parameters

`lambda numeric(1)`
: Value used for sampling the lambda for each worker from an exponential distribution.

`rate numeric(1)`
: Rate of the exponential decay.

`period integer(1)`
: Period of the exponential decay.

`initial_design data.table::data.table()`
: Initial design of the optimization. If `NULL`, a design of size `design_size` is generated with the specified `design_function`. Default is `NULL`.

`design_size integer(1)`
: Size of the initial design if it is to be generated. Default is `100`.

`design_function character(1)`
: Sampling function to generate the initial design. Can be `random` paradox::generate_design_random, `lhs` paradox::generate_design_lhs, or `sobol` paradox::generate_design_sobol. Default is `sobol`.

`n_workers integer(1)`
: Number of parallel workers. If `NULL`, all rush workers specified via `rush::rush_plan()` are used. Default is `NULL`.

Super classes

`bbotk::Optimizer` -> `bbotk::OptimizerAsync` -> `mlr3mbo::OptimizerAsyncMbo` -> `OptimizerADBO`

Methods

Public methods:

- `OptimizerADBO$new()`
- `OptimizerADBO$optimize()`
- `OptimizerADBO$clone()`

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

Usage:

```
OptimizerADBO$new()
```

Method `optimize()`**:** Performs the optimization on an bbotk::OptimInstanceAsyncSingleCrit until termination. The single evaluations will be written into the bbotk::ArchiveAsync. The result will be written into the instance object.

Usage:

```
OptimizerADBO$optimize(inst)
```

Arguments:

`inst` (bbotk::OptimInstanceAsyncSingleCrit).

Returns: `data.table::data.table()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

The lambda parameter of the confidence bound acquisition function controls the trade-off between exploration and exploitation. A large lambda value leads to more exploration, while a small lambda value leads to more exploitation. The initial lambda value of the acquisition function used on each worker is drawn from an exponential distribution with rate `1 / lambda`. ADBO can use periodic exponential decay to reduce lambda periodically for a given time step `t` with the formula `lambda * exp(-rate * (t %% period))`. The SurrogateLearner is configured to use a random forest and the AcqOptimizer is a random search with a batch size of 1000 and a budget of 10000 evaluations.

References

- Egelé, Romain, Guyon, Isabelle, Vishwanath, Venkatram, Balaprakash, Prasanna (2023). "Asynchronous Decentralized Bayesian Optimization for Large Scale Hyperparameter Optimization." In *2023 IEEE 19th International Conference on e-Science (e-Science)*, 1–10.

Examples

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

  if (redis_available()) {

    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 = OptimInstanceAsyncSingleCrit$new(
      objective = objective,
      terminator = trm("evals", n_evals = 10))

    mirai::daemons(2)
    rush::rush_plan(n_workers=2, worker_type = "mirai")

    optimizer = opt("adbo", design_size = 4, n_workers = 2)

    optimizer$optimize(instance)
    mirai::daemons(0)
  } else {
   message("Redis server is not available.\nPlease set up Redis prior to running the example.")
  }
}
```

---

mlr_optimizers_async_mbo

Asynchronous Model Based Optimization

Description

`OptimizerAsyncMbo` class that implements Asynchronous Model Based Optimization (AMBO). AMBO starts multiple sequential MBO runs on different workers. The worker communicate asynchronously through a shared archive relying on the **rush** package. The optimizer follows a modular layout in which the surrogate model, acquisition function, and acquisition optimizer can be changed. The SurrogateLearner will impute missing values due to pending evaluations. A stochastic AcqFunction, e.g., AcqFunctionStochasticEI or AcqFunctionStochasticCB is used to create varying versions of the acquisition function on each worker, promoting different exploration-exploitation trade-offs. The AcqOptimizer class remains consistent with the one used in synchronous MBO.

In contrast to OptimizerMbo, no loop_function can be specified that determines the AMBO flavor as `OptimizerAsyncMbo` simply relies on a surrogate update, acquisition function update and acquisition function optimization step as an internal loop.

Currently, only single-objective optimization is supported and `OptimizerAsyncMbo` is considered an experimental feature and API might be subject to changes.

Note that in general the SurrogateLearner 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. It is therefore recommended to manually inspect the SurrogateLearner 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 SurrogateLearner could not be performed successfully, a warning will be logged.

By specifying a ResultAssigner, one can alter how the final result is determined after optimization, e.g., simply based on the evaluations logged in the archive ResultAssignerArchive or based on the Surrogate via ResultAssignerSurrogate.

Archive

The bbotk::ArchiveAsync holds the following additional columns that are specific to AMBO 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.

If the bbotk::ArchiveAsync does not contain any evaluations prior to optimization, an initial design is needed. If the `initial_design` parameter is specified to be a `data.table`, this data will be used. Otherwise, if it is `NULL`, an initial design of size `design_size` will be generated based on the `generate_design` sampling function. See also the parameters below.

Parameters

`initial_design data.table::data.table()`
: Initial design of the optimization. If `NULL`, a design of size `design_size` is generated with the specified `design_function`. Default is `NULL`.

`design_size integer(1)`
: Size of the initial design if it is to be generated. Default is `100`.

`design_function character(1)`
: Sampling function to generate the initial design. Can be `random` paradox::generate_design_random, `lhs` paradox::generate_design_lhs, or `sobol` paradox::generate_design_sobol. Default is `sobol`.

`n_workers integer(1)`
: Number of parallel workers. If `NULL`, all rush workers specified via `rush::rush_plan()` are used. Default is `NULL`.

Conditions

During optimization, errors are caught and re-thrown as structured conditions that inherit from `Mlr3ErrorMbo` and `Mlr3Error` (defined in **mlr3misc**). Loop functions catch `Mlr3ErrorMbo` conditions and fall back to proposing a randomly sampled point.

The following condition classes are used:

`Mlr3ErrorMbo` Base class for all MBO-specific error conditions.

`Mlr3ErrorMboSurrogateUpdate` Raised by Surrogate`$update()` when the surrogate model fails to update (requires `catch_errors = TRUE`).

`Mlr3ErrorMboAcqOptimizer` Raised by AcqOptimizer`$optimize()` when the acquisition function optimization fails (requires `catch_errors = TRUE`).

`Mlr3ErrorMboRandomInterleave` Raised by loop functions to trigger random interleaving.

`Mlr3ErrorMboSurrogateUpdate` and `Mlr3ErrorMboAcqOptimizer` conditions are logged at the `"warn"` level and include the original error as a parent condition. All conditions can be constructed directly via the helper functions `error_surrogate_update()`, `error_acq_optimizer()`, and `error_random_interleave()`.

Super classes

`bbotk::Optimizer` -> `bbotk::OptimizerAsync` -> `OptimizerAsyncMbo`

Active bindings

`surrogate` (Surrogate | `NULL`)
: The surrogate.

`acq_function` (AcqFunction | `NULL`)
: The acquisition function.

`acq_optimizer` (AcqOptimizer | `NULL`)
: The acquisition function optimizer.

`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, e.g., 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:

- `OptimizerAsyncMbo$new()`
- `OptimizerAsyncMbo$print()`
- `OptimizerAsyncMbo$reset()`
- `OptimizerAsyncMbo$optimize()`
- `OptimizerAsyncMbo$clone()`

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

If `surrogate` is `NULL` and the `acq_function$surrogate` field is populated, this SurrogateLearner is used. Otherwise, `default_surrogate(instance)` is used. If `acq_function` is `NULL` and the `acq_optimizer$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::ArchiveAsync of the current bbotk::OptimInstanceAsyncSingleCrit to be optimized.

For more information on default values for `surrogate`, `acq_function`, `acq_optimizer` and `result_assigner`, see `?mbo_defaults`.

Usage:

```
OptimizerAsyncMbo$new(
  id = "async_mbo",
  surrogate = NULL,
  acq_function = NULL,
  acq_optimizer = NULL,
  result_assigner = NULL,
  param_set = NULL,
  label = "Asynchronous Model Based Optimization",
  man = "mlr3mbo::OptimizerAsyncMbo"
)
```

Arguments:

`id` (`character(1)`)
: Identifier for the new instance.

`surrogate` (Surrogate | `NULL`)
: The surrogate.

`acq_function` (AcqFunction | `NULL`)
The acquisition function.

`acq_optimizer` (AcqOptimizer | `NULL`)
The acquisition function optimizer.

`result_assigner` (ResultAssigner | `NULL`)
The result assigner.

`param_set` (paradox::ParamSet)
Set of control parameters.

`label` (`character(1)`)
Label for this object. Can be used in tables, plot and text output instead of the ID.

`man` (`character(1)`)
String in the format `[pkg]::[topic]` pointing to a manual page for this object. The referenced help package can be opened via method `$help()`.

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

Usage:

```
OptimizerAsyncMbo$print()
```

Returns: (`character()`).

Method `reset()`**:** Reset the optimizer. Sets the following fields to `NULL`: `surrogate`, `acq_function`, `acq_optimizer`,`result_assigner` Resets parameter values `design_size` and `design_function` to their defaults.

Usage:

```
OptimizerAsyncMbo$reset()
```

Method `optimize()`**:** Performs the optimization on an bbotk::OptimInstanceAsyncSingleCrit until termination. The single evaluations will be written into the bbotk::ArchiveAsync. The result will be written into the instance object.

Usage:

```
OptimizerAsyncMbo$optimize(inst)
```

Arguments:

`inst` (bbotk::OptimInstanceAsyncSingleCrit).

Returns: `data.table::data.table()`

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

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

  if (redis_available()) {

    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 = OptimInstanceAsyncSingleCrit$new(
      objective = objective,
      terminator = trm("evals", n_evals = 10))

    mirai::daemons(2)
    rush::rush_plan(n_workers=2, worker_type = "mirai")

    optimizer = opt("async_mbo", design_size = 4, n_workers = 2)

    optimizer$optimize(instance)
    mirai::daemons(0)
  } else {
   message("Redis server is not available.\nPlease set up Redis prior to running the example.")
  }
}
```

---

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::OptimInstanceBatch 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.

By specifying a ResultAssigner, one can alter how the final result is determined after optimization, e.g., simply based on the evaluations logged in the archive ResultAssignerArchive or based on the Surrogate via ResultAssignerSurrogate.

Archive

The bbotk::ArchiveBatch 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.

Conditions

During optimization, errors are caught and re-thrown as structured conditions that inherit from `Mlr3ErrorMbo` and `Mlr3Error` (defined in **mlr3misc**). Loop functions catch `Mlr3ErrorMbo` conditions and fall back to proposing a randomly sampled point.

The following condition classes are used:

`Mlr3ErrorMbo` Base class for all MBO-specific error conditions.

`Mlr3ErrorMboSurrogateUpdate` Raised by Surrogate`$update()` when the surrogate model fails to update (requires `catch_errors = TRUE`).

`Mlr3ErrorMboAcqOptimizer` Raised by AcqOptimizer`$optimize()` when the acquisition function optimization fails (requires `catch_errors = TRUE`).

`Mlr3ErrorMboRandomInterleave` Raised by loop functions to trigger random interleaving.

`Mlr3ErrorMboSurrogateUpdate` and `Mlr3ErrorMboAcqOptimizer` conditions are logged at the `"warn"` level and include the original error as a parent condition. All conditions can be constructed directly via the helper functions `error_surrogate_update()`, `error_acq_optimizer()`, and `error_random_interleave()`.

Super classes

`bbotk::Optimizer` -> `bbotk::OptimizerBatch` -> `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, e.g., 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$reset()`
- `OptimizerMbo$optimize()`
- `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::ArchiveBatch of the current bbotk::OptimInstanceBatch to be optimized.

For more information on default values for `loop_function`, `surrogate`, `acq_function`, `acq_optimizer` and `result_assigner`, 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 `optimize()`**:** Performs the optimization and writes optimization result into bbotk::OptimInstanceBatch. The optimization result is returned but the complete optimization path is stored in bbotk::ArchiveBatch of bbotk::OptimInstanceBatch.

Usage:

```
OptimizerMbo$optimize(inst)
```

Arguments:

`inst` (bbotk::OptimInstanceBatch).

Returns: data.table::data.table.

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 = OptimInstanceBatchSingleCrit$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 = OptimInstanceBatchMultiCrit$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_output_trafos` *Dictionary of Output Transformations*

Description

A simple mlr3misc::Dictionary storing objects of class OutputTrafo. Each output transformation has an associated help page, see `mlr_output_trafos[id]`.

For a more convenient way to retrieve and construct an output trafo, see `ot()`.

Format

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

Methods

See mlr3misc::Dictionary.

See Also

Sugar function: `ot()`

Other Dictionary: `mlr_acqfunctions`, `mlr_acqoptimizers`, `mlr_input_trafos`, `mlr_loop_functions`, `mlr_result_assigners`

Other Output Transformation: `OutputTrafo`, `OutputTrafoLog`, `OutputTrafoStandardize`

Examples

```
library(data.table)
as.data.table(mlr_output_trafos)
ot("standardize")
```

mlr_result_assigners *Dictionary of Result Assigners*

Description

A simple mlr3misc::Dictionary storing objects of class ResultAssigner. Each result assigner has an associated help page, see `mlr_result_assigners_[id]`.

For a more convenient way to retrieve and construct a result assigner, see `ras()`.

Format

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

Methods

See mlr3misc::Dictionary.

See Also

Sugar function: `ras()`

Other Dictionary: `mlr_acqfunctions`, `mlr_acqoptimizers`, `mlr_input_trafos`, `mlr_loop_functions`, `mlr_output_trafos`

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

Examples

```
library(data.table)
as.data.table(mlr_result_assigners)
ras("archive")
```

mlr_result_assigners_archive *Result Assigner Based on the Archive*

Description

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

Super class

`mlr3mbo::ResultAssigner` -> `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 `requireNamespace()`.

Methods

Public methods:

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

Method `new()`**:** Creates a new instance of this 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::OptimInstanceBatchSingleCrit | bbotk::OptimInstanceBatchMultiCrit | bbotk::OptimInstanceAsyncSi | bbotk::OptimInstanceAsyncMultiCrit)
: 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")
```

`mlr_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::OptimInstanceBatchMultiCrit or bbotk::OptimInstanceAsyncMultiCrit the SurrogateLearnerCollection must use as many learners as there are objective functions.

Super class

`mlr3mbo::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::OptimInstanceBatchSingleCrit | bbotk::OptimInstanceBatchMultiCrit | bbotk::OptimInstanceAsyncSi
| bbotk::OptimInstanceAsyncMultiCrit)
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

```
result_assigner = ras("surrogate")
```

`mlr_tuners_adbo` *TunerAsync using Asynchronous Decentralized Bayesian Optimization*

Description

`TunerADBO` class that implements Asynchronous Decentralized Bayesian Optimization (ADBO). ADBO is a variant of Asynchronous Model Based Optimization (AMBO) that uses AcqFunctionStochasticCB with exponential lambda decay. This is a minimal interface internally passing on to OptimizerAsyncMbo. For additional information and documentation see OptimizerAsyncMbo.

Currently, only single-objective optimization is supported and `TunerADBO` is considered an experimental feature and API might be subject to changes.

Parameters

`initial_design data.table::data.table()`
: Initial design of the optimization. If `NULL`, a design of size `design_size` is generated with the specified `design_function`. Default is `NULL`.

`design_size integer(1)`
: Size of the initial design if it is to be generated. Default is `100`.

`design_function character(1)`
: Sampling function to generate the initial design. Can be `random` paradox::generate_design_random, `lhs` paradox::generate_design_lhs, or `sobol` paradox::generate_design_sobol. Default is `sobol`.

`n_workers integer(1)`
: Number of parallel workers. If `NULL`, all rush workers specified via `rush::rush_plan()` are used. Default is `NULL`.

Super classes

`mlr3tuning::Tuner` -> `mlr3tuning::TunerAsync` -> `mlr3tuning::TunerAsyncFromOptimizerAsync` -> `TunerADBO`

Active bindings

`surrogate` (Surrogate | `NULL`)
The surrogate.

`acq_function` (AcqFunction | `NULL`)
The acquisition function.

`acq_optimizer` (AcqOptimizer | `NULL`)
The acquisition function optimizer.

`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, e.g., 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:

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

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

Usage:

```
TunerADBO$new()
```

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

Usage:

```
TunerADBO$print()
```

Returns: (`character()`).

Method `reset()`**:** Reset the tuner. Sets the following fields to `NULL`: `surrogate`, `acq_function`, `acq_optimizer`, `result_assigner` Resets parameter values `design_size` and `design_function` to their defaults.

Usage:

```
TunerADBO$reset()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

- Egelé, Romain, Guyon, Isabelle, Vishwanath, Venkatram, Balaprakash, Prasanna (2023). "Asynchronous Decentralized Bayesian Optimization for Large Scale Hyperparameter Optimization." In *2023 IEEE 19th International Conference on e-Science (e-Science)*, 1–10.

Examples

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

    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 = TuningInstanceAsyncSingleCrit$new(
      task = task,
      learner = learner,
      resampling = resampling,
      measure = measure,
      terminator = trm("evals", n_evals = 10))

    mirai::daemons(2)
    rush::rush_plan(n_workers=2, worker_type = "mirai")

    tnr("adbo", design_size = 4, n_workers = 2)$optimize(instance)
    mirai::daemons(0)
  } else {
   message("Redis server is not available.\nPlease set up Redis prior to running the example.")
  }
}
```

`mlr_tuners_async_mbo` *TunerAsync using Asynchronous Model Based Optimization*

Description

`TunerAsyncMbo` class that implements Asynchronous Model Based Optimization (AMBO). This is a minimal interface internally passing on to OptimizerAsyncMbo. For additional information and documentation see OptimizerAsyncMbo.

Currently, only single-objective optimization is supported and `TunerAsyncMbo` is considered an experimental feature and API might be subject to changes.

Parameters

`initial_design data.table::data.table()`
: Initial design of the optimization. If `NULL`, a design of size `design_size` is generated with the specified `design_function`. Default is `NULL`.

`design_size integer(1)`
: Size of the initial design if it is to be generated. Default is `100`.

`design_function character(1)`
: Sampling function to generate the initial design. Can be `random` paradox::generate_design_random, `lhs` paradox::generate_design_lhs, or `sobol` paradox::generate_design_sobol. Default is `sobol`.

`n_workers integer(1)`
: Number of parallel workers. If `NULL`, all rush workers specified via `rush::rush_plan()` are used. Default is `NULL`.

Super classes

`mlr3tuning::Tuner` -> `mlr3tuning::TunerAsync` -> `mlr3tuning::TunerAsyncFromOptimizerAsync` -> `TunerAsyncMbo`

Active bindings

`surrogate` (Surrogate | `NULL`)
: The surrogate.

`acq_function` (AcqFunction | `NULL`)
: The acquisition function.

`acq_optimizer` (AcqOptimizer | `NULL`)
: The acquisition function optimizer.

`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, e.g., 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:

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

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

Usage:

```
TunerAsyncMbo$new(
  surrogate = NULL,
  acq_function = NULL,
  acq_optimizer = NULL,
  param_set = NULL
)
```

Arguments:

`surrogate` (Surrogate | `NULL`)
: The surrogate.

`acq_function` (AcqFunction | `NULL`)
: The acquisition function.

`acq_optimizer` (AcqOptimizer | `NULL`)
: The acquisition function optimizer.

`param_set` (paradox::ParamSet)
: Set of control parameters.

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

Usage:

```
TunerAsyncMbo$print()
```

Returns: (`character()`).

Method `reset()`**:** Reset the tuner. Sets the following fields to `NULL`: `surrogate`, `acq_function`, `acq_optimizer`, `result_assigner` Resets parameter values `design_size` and `design_function` to their defaults.

Usage:

```
TunerAsyncMbo$reset()
```

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

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

  if (redis_available()) {

    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 = TuningInstanceAsyncSingleCrit$new(
      task = task,
      learner = learner,
      resampling = resampling,
      measure = measure,
      terminator = trm("evals", n_evals = 10))

    mirai::daemons(2)
    rush::rush_plan(n_workers=2, worker_type = "mirai")

    tnr("async_mbo", design_size = 4, n_workers = 2)$optimize(instance)
    mirai::daemons(0)
  } else {
   message("Redis server is not available.\nPlease set up Redis prior to running the example.")
  }
}
```

`mlr_tuners_mbo` *TunerBatch 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::TunerBatch -> mlr3tuning::TunerBatchFromOptimizerBatch -> 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_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, e.g., 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:

- 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`, `acq_optimizer`, and `result_assigner`, 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_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:

```
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:

```
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 = TuningInstanceBatchSingleCrit$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 = TuningInstanceBatchMultiCrit$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)
}
```

---

ot *Syntactic Sugar Output Trafo Construction*

Description

This function complements mlr_output_trafos with functions in the spirit of `mlr_sugar` from **mlr3**.

Usage

```
ot(.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

OutputTrafo

Examples

```
ot("standardize")
```

OutputTrafo *Output Transformation Base Class*

Description

Abstract output transformation class.

An output transformation can be used within a Surrogate to perform a transformation of the target variable(s).

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()`.

`state` (named `list()` | `NULL`)
: List of meta information regarding the parameters and their state.

`cols_y` (`character()` | `NULL`)
: Column ids of target variables that should be transformed.

`max_to_min` (`-1` | `1`)
: Multiplicative factor to correct for minimization or maximization.

`invert_posterior` (`logical(1)`)
: Should the posterior predictive distribution be inverted when used within a SurrogateLearner or SurrogateLearnerCollection?

Methods

Public methods:

- `OutputTrafo$new()`
- `OutputTrafo$update()`
- `OutputTrafo$transform()`
- `OutputTrafo$inverse_transform_posterior()`
- `OutputTrafo$inverse_transform()`
- `OutputTrafo$format()`
- `OutputTrafo$print()`
- `OutputTrafo$clone()`

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

Usage:

```
OutputTrafo$new(invert_posterior, label = NA_character_, man = NA_character_)
```

Arguments:

`invert_posterior` (`logical(1)`)
Should the posterior predictive distribution be inverted when used within a SurrogateLearner or SurrogateLearnerCollection?

`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()`**:** Learn the transformation based on observed data and update parameters in `$state`. Must be implemented by subclasses.

Usage:

```
OutputTrafo$update(ydt)
```

Arguments:

`ydt` (`data.table::data.table()`)
Data. One row per observation with at least columns `$cols_y`.

Method `transform()`**:** Perform the transformation. Must be implemented by subclasses.

Usage:

```
OutputTrafo$transform(ydt)
```

Arguments:

`ydt` (`data.table::data.table()`)
Data. One row per observation with at least columns `$cols_y`.

Returns: `data.table::data.table()` with the transformation applied to the columns `$cols_y`.

Method `inverse_transform_posterior()`**:** Perform the inverse transformation on a posterior predictive distribution characterized by the first and second moment. Must be implemented by subclasses.

Usage:

```
OutputTrafo$inverse_transform_posterior(pred)
```

Arguments:

`pred` (`data.table::data.table()`)
Data. One row per observation characterizing a posterior predictive distribution with the columns `mean` and `se`.

Returns: `data.table::data.table()` with the inverse transformation applied to the columns `mean` and `se`.

Method `inverse_transform()`**:** Perform the inverse transformation. Must be implemented by subclasses.

Usage:

```
OutputTrafo$inverse_transform(ydt)
```

Arguments:

`ydt` (`data.table::data.table()`)
Data. One row per observation with at least columns `$cols_y`.

Returns: `data.table::data.table()` with the inverse transformation applied to the columns `$cols_y`.

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

Usage:

```
OutputTrafo$format()
```

Returns: (`character(1)`).

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

Usage:

```
OutputTrafo$print()
```

Returns: (`character()`).

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Output Transformation: `OutputTrafoLog`, `OutputTrafoStandardize`, `mlr_output_trafos`

`OutputTrafoLog` *Output Transformation Log*

Description

Output transformation that takes the logarithm after min-max scaling to $(0, 1)$.

Super class

`mlr3mbo::OutputTrafo` -> `OutputTrafoLog`

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 `requireNamespace()`.

Methods

Public methods:

- `OutputTrafoLog$new()`
- `OutputTrafoLog$update()`
- `OutputTrafoLog$transform()`
- `OutputTrafoLog$inverse_transform_posterior()`
- `OutputTrafoLog$inverse_transform()`
- `OutputTrafoLog$clone()`

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

Usage:

```
OutputTrafoLog$new(invert_posterior = FALSE)
```

Arguments:

`invert_posterior (logical(1))`
Should the posterior predictive distribution be inverted when used within a SurrogateLearner or SurrogateLearnerCollection? Default is `FALSE`.

Method `update()`**:** Learn the transformation based on observed data and update parameters in `$state`.

Usage:

```
OutputTrafoLog$update(ydt)
```

Arguments:

`ydt (data.table::data.table())`
Data. One row per observation with columns `$cols_y`.

Method `transform()`**:** Perform the transformation.

Usage:

```
OutputTrafoLog$transform(ydt)
```

Arguments:

`ydt` (`data.table::data.table()`)
Data. One row per observation with at least columns `$cols_y`.

Returns: `data.table::data.table()` with the transformation applied to the columns `$cols_y`.

Method `inverse_transform_posterior()`**:** Perform the inverse transformation on a posterior predictive distribution characterized by the first and second moment.

Usage:

```
OutputTrafoLog$inverse_transform_posterior(pred)
```

Arguments:

`pred` (`data.table::data.table()`)
Data. One row per observation characterizing a posterior predictive distribution with the columns `mean` and `se`. Can also be a named list of `data.table::data.table()` with posterior predictive distributions for multiple targets corresponding to (`cols_y`).

Returns: `data.table::data.table()` with the inverse transformation applied to the columns `mean` and `se`. In the case of the input being a named list of `data.table::data.table()`, the output will be a named list of `data.table::data.table()` with the inverse transformation applied to the columns `mean` and `se`.

Method `inverse_transform()`**:** Perform the inverse transformation.

Usage:

```
OutputTrafoLog$inverse_transform(ydt)
```

Arguments:

`ydt` (`data.table::data.table()`)
Data. One row per observation with at least columns `$cols_y`.

Returns: `data.table::data.table()` with the inverse transformation applied to the columns `$cols_y`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Output Transformation: `OutputTrafo`, `OutputTrafoStandardize`, `mlr_output_trafos`

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 = OptimInstanceBatchSingleCrit$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()

  output_trafo = ot("log", invert_posterior = TRUE)

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

  surrogate$update()

  surrogate$output_trafo$state

  surrogate$predict(data.table(x = c(-1, 0, 1)))

  surrogate$output_trafo$invert_posterior = FALSE

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

OutputTrafoStandardize

Output Transformation Standardization

Description

Output transformation that performs standardization to zero mean and unit variance.

Super class

`mlr3mbo::OutputTrafo` -> `OutputTrafoStandardize`

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 `requireNamespace()`.

Methods

Public methods:

- `OutputTrafoStandardize$new()`
- `OutputTrafoStandardize$update()`
- `OutputTrafoStandardize$transform()`
- `OutputTrafoStandardize$inverse_transform_posterior()`
- `OutputTrafoStandardize$inverse_transform()`
- `OutputTrafoStandardize$clone()`

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

Usage:

```
OutputTrafoStandardize$new(invert_posterior = TRUE)
```

Arguments:

`invert_posterior (logical(1))`
: Should the posterior predictive distribution be inverted when used within a SurrogateLearner or SurrogateLearnerCollection? Default is `TRUE`.

Method `update()`**:** Learn the transformation based on observed data and update parameters in `$state`.

Usage:

```
OutputTrafoStandardize$update(ydt)
```

Arguments:

`ydt (data.table::data.table())`
: Data. One row per observation with columns `$cols_y`.

Method `transform()`**:** Perform the transformation.

Usage:

```
OutputTrafoStandardize$transform(ydt)
```

Arguments:

`ydt (data.table::data.table())`
: Data. One row per observation with at least columns `$cols_y`.

Returns: `data.table::data.table()` with the transformation applied to the columns `$cols_y`.

Method `inverse_transform_posterior()`**:** Perform the inverse transformation on a posterior predictive distribution characterized by the first and second moment.

Usage:

```
OutputTrafoStandardize$inverse_transform_posterior(pred)
```

Arguments:

`pred` (`data.table::data.table()`)
: Data. One row per observation characterizing a posterior predictive distribution with the columns `mean` and `se`. Can also be a named list of `data.table::data.table()` with posterior predictive distributions for multiple targets corresponding to (`cols_y`).

Returns: `data.table::data.table()` with the inverse transformation applied to the columns `mean` and `se`. In the case of the input being a named list of `data.table::data.table()`, the output will be a named list of `data.table::data.table()` with the inverse transformation applied to the columns `mean` and `se`.

Method `inverse_transform()`**:** Perform the inverse transformation.

Usage:

```
OutputTrafoStandardize$inverse_transform(ydt)
```

Arguments:

`ydt` (`data.table::data.table()`)
: Data. One row per observation with at least columns `$cols_y`.

Returns: `data.table::data.table()` with the inverse transformation applied to the columns `$cols_y`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

See Also

Other Output Transformation: `OutputTrafo`, `OutputTrafoLog`, `mlr_output_trafos`

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 = OptimInstanceBatchSingleCrit$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()

  output_trafo = ot("standardize", invert_posterior = TRUE)

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

  surrogate$update()

  surrogate$output_trafo$state

  surrogate$predict(data.table(x = c(-1, 0, 1)))

  surrogate$output_trafo$invert_posterior = FALSE

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

`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")
```

redis_available *Check if Redis Server is Available*

Description

Attempts to establish a connection to a Redis server using the **redux** package and sends a `PING` command. Returns `TRUE` if the server is available and responds appropriately, `FALSE` otherwise.

Usage

```
redis_available()
```

Value

`(logical(1))`

Examples

```
if (redis_available()) {
  # Proceed with code that requires Redis
  message("Redis server is available.")
} else {
  message("Redis server is not available.")
}
```

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:

```
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:

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

Arguments:

`instance` (bbotk::OptimInstanceBatchSingleCrit | bbotk::OptimInstanceBatchMultiCrit | bbotk::OptimInstanceAsyncSi
| bbotk::OptimInstanceAsyncMultiCrit)
The bbotk::OptimInstance the final result should be assigned to.

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

Usage:

```
ResultAssigner$format()
```

Returns: (`character(1)`).

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

Usage:

```
ResultAssigner$print()
```

Returns: (`character()`).

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

Usage:

```
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`

srlrn *Syntactic Sugar Surrogate Construction*

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

```
srlrn(
  learner,
  input_trafo = NULL,
  output_trafo = NULL,
  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.
`input_trafo`	(`NULL` | InputTrafo) Input transformation. Can also be `NULL`.'
`output_trafo`	(`NULL` | OutputTrafo) Output transformation. Can also be `NULL`.
`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

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

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

`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.

`archive_is_async` ('bool(1)")
: Whether the bbotk::Archive is an asynchronous one.

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

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

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

`param_set` (paradox::ParamSet)
: Set of hyperparameters.

`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$reset()`
- `Surrogate$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()` and `private.update_async()`.

Usage:

```
Surrogate$update()
```

Returns: `NULL`.

Method `reset()`**:** Reset the surrogate model. Subclasses must implement `private$.reset()`.

Usage:

```
Surrogate$reset()
```

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()
```

Returns: (`character(1)`).

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

`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`.

`impute_method character(1)`
Method to impute missing values in the case of updating on an asynchronous bbotk::ArchiveAsync with pending evaluations. Can be `"mean"` to use mean imputation or `"random"` to sample values uniformly at random between the empirical minimum and maximum. Default is `"random"`.

Super class

`mlr3mbo::Surrogate` -> `SurrogateLearner`

Public fields

`learner` (mlr3::LearnerRegr)
: mlr3::LearnerRegr wrapped as a surrogate model.

`input_trafo` (InputTrafo)
: Input transformation.

`output_trafo` (OutputTrafo)
: Output transformation.

Active bindings

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

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

`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"`.

`output_trafo_must_be_considered` (logical(1))
: Whether a transformation has been applied to the target variable that has not been inverted during prediction.

Methods

Public methods:

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

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

Usage:

```
SurrogateLearner$new(
  learner,
  input_trafo = NULL,
```

```
  output_trafo = NULL,
  archive = NULL,
  cols_x = NULL,
  col_y = NULL
)
```

Arguments:

`learner` (mlr3::LearnerRegr).

`input_trafo` (InputTrafo | `NULL`). InputTrafo to be applied.

`output_trafo` (OutputTrafo | `NULL`). OutputTrafo to be applied.

`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:

```
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 = OptimInstanceBatchSingleCrit$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
}
```

`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 `cols_y`. Note that redundant mlr3::LearnerRegr must be deep clones.

Parameters

`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`.

`impute_method character(1)`
: Method to impute missing values in the case of updating on an asynchronous bbotk::ArchiveAsync with pending evaluations. Can be `"mean"` to use mean imputation or `"random"` to sample values uniformly at random between the empirical minimum and maximum. Default is `"random"`.

Super class

`mlr3mbo::Surrogate` -> `SurrogateLearnerCollection`

Public fields

`learner` (list of mlr3::LearnerRegr)
: List of mlr3::LearnerRegr wrapped as surrogate models.

`input_trafo` (InputTrafo)
: Input transformation.

`output_trafo` (OutputTrafo)
: Output transformation.

Active bindings

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

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

`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"`.

`output_trafo_must_be_considered` (logical(1))
: Whether a transformation has been applied to the target variable that has not been inverted during prediction.

Methods

Public methods:

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

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

Usage:

```
SurrogateLearnerCollection$new(
  learners,
  input_trafo = NULL,
  output_trafo = NULL,
  archive = NULL,
  cols_x = NULL,
  cols_y = NULL
)
```

Arguments:

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

`input_trafo` (InputTrafo | `NULL`). InputTrafo to be applied.

`output_trafo` (OutputTrafo | `NULL`). OutputTrafo to be applied.

`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.

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

Usage:

```
SurrogateLearnerCollection$predict(xdt)
```

Arguments:

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

Returns: named list of `data.table::data.table()` 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 = OptimInstanceBatchMultiCrit$new(
    objective = objective,
```

```
  terminator = trm("evals", n_evals = 5))

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

instance$eval_batch(xdt)

learner1 = default_gp()

learner2 = default_rf()

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

surrogate$update()

surrogate$learner

surrogate$learner[["y1"]]$model

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

Index