mlrMBO

Toolbox for Bayesian Optimization and Model-Based Optimization in R

Jakob Richter [1]   Bernd Bischl [2]   Jakob Bossek [3]   Michel Lang [1]

July 3, 2018

[1]TU Dortmund, Germany

[2]LMU Munich, Germany

[3]WWU Münster, Germany

Model-Based Optimization

Optimization Problem:

$$y = f(\boldsymbol{x}) \ , \quad f : \mathbb{X} \to \mathbb{R}$$

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x} \in \mathbb{X}} f(\boldsymbol{x})$$

Model-Based Optimization

Optimization Problem:

$$y = f(\boldsymbol{x})\ , \quad f : \mathbb{X} \to \mathbb{R}$$

$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in \mathbb{X}}{\arg\max}\, f(\boldsymbol{x})$$

But:

- $f'(\boldsymbol{x})$ unknown
- often: $\mathbb{X} \nsubseteq \mathbb{R}^d$ but $[-10, 10]^3 \times \{A, B, C\} \times \dots$
- also: $y = f(\boldsymbol{x}) + \varepsilon(\boldsymbol{x})$
- $f(\boldsymbol{x}^*)$ unknown

Model-Based Optimization

Optimization Problem:

$$y = f(\boldsymbol{x}) \ , \quad f : \mathbb{X} \to \mathbb{R}$$
$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in \mathbb{X}}{\arg\max} \, f(\boldsymbol{x})$$

But:

- $f'(\boldsymbol{x})$ unknown
- often: $\mathbb{X} \nsubseteq \mathbb{R}^d$ but $[-10, 10]^3 \times \{A, B, C\} \times \ldots$
- also: $y = f(\boldsymbol{x}) + \varepsilon(\boldsymbol{x})$
- $f(\boldsymbol{x}^*)$ unknown

Main challenge:

⌛ Evaluation of $f(\boldsymbol{x})$ can take > 30 minutes.

Model-Based Optimization

Optimization Problem:

$$y = f(\boldsymbol{x}) \ , \quad f : \mathbb{X} \to \mathbb{R}$$
$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x} \in \mathbb{X}} f(\boldsymbol{x})$$

But:

- $f'(\boldsymbol{x})$ unknown
- often: $\mathbb{X} \nsubseteq \mathbb{R}^d$ but $[-10, 10]^3 \times \{A, B, C\} \times \dots$
- also: $y = f(\boldsymbol{x}) + \varepsilon(\boldsymbol{x})$
- $f(\boldsymbol{x}^*)$ unknown

Main challenge:

Evaluation of $f(\boldsymbol{x})$ can take > 30 minutes.

Therefore: ~~Gradient-, (Quasi-)Newton-, Evolutionary Methods~~

Model-Based Optimization

No additional information for f.
Only possibility: Selective evaluation of $f(\boldsymbol{x})$ and acquiring knowledge of evaluated points $(\boldsymbol{x}, y)$.

Wanted: Strategy to select $\boldsymbol{x}$ so that we get to the optimum quickly.

Model-Based Optimization

No additional information for f.
Only possibility: Selective evaluation of $f(\boldsymbol{x})$ and acquiring knowledge of evaluated points $(\boldsymbol{x}, y)$.

- Wanted: Strategy to select $\boldsymbol{x}$ so that we get to the optimum quickly.
- Idea: Evaluate $f(\boldsymbol{x})$ for some $\boldsymbol{x}$ and then fit a regression model $\hat{f}(\boldsymbol{x})$.

Model-Based Optimization

No additional information for f.
Only possibility: Selective evaluation of $f(\boldsymbol{x})$ and acquiring knowledge of evaluated points $(\boldsymbol{x}, y)$.

- Wanted: Strategy to select $\boldsymbol{x}$ so that we get to the optimum quickly.
- Idea: Evaluate $f(\boldsymbol{x})$ for some $\boldsymbol{x}$ and then fit a regression model $\hat{f}(\boldsymbol{x})$.
- Hope: Maximum of $\hat{f}(\boldsymbol{x})$ is close to maximum of $f(\boldsymbol{x})$.

Model-Based Optimization

No additional information for f.
Only possibility: Selective evaluation of $f(\boldsymbol{x})$ and acquiring knowledge of evaluated points $(\boldsymbol{x}, y)$.

- Wanted: Strategy to select $\boldsymbol{x}$ so that we get to the optimum quickly.
- Idea: Evaluate $f(\boldsymbol{x})$ for some $\boldsymbol{x}$ and then fit a regression model $\hat{f}(\boldsymbol{x})$.
- Hope: Maximum of $\hat{f}(\boldsymbol{x})$ is close to maximum of $f(\boldsymbol{x})$.
- Why the detour? We can usually calculate the maximum of $\hat{f}(\boldsymbol{x})$ in a few seconds.

Motivation: Hyperparameter Tuning

MBO in Machine Learning

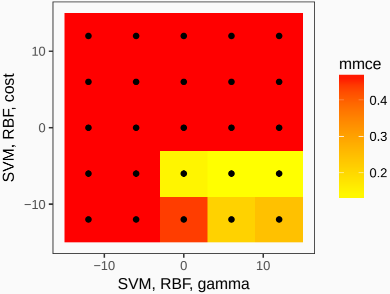
$$f(\boldsymbol{x}) = y$$

$\boldsymbol{x}$: hyperparameter setting

y : Prediction performance (evaluated by resampling)

- Still common practice: grid search
  For a SVM it might look like:
  - $C \in (2^{-12}, 2^{-10}, 2^{-8}, \ldots, 2^{8}, 2^{10}, 2^{12})$
  - $\gamma \in (2^{-12}, 2^{-10}, 2^{-8}, \ldots, 2^{8}, 2^{10}, 2^{12})$
  - Evaluate all $13^2 = 169$ combinations $C \times \gamma$
- Bad because:
  - optimum might be "off the grid"
  - lots of evaluations in bad areas
  - lots of costly evaluations
- How bad? $\hookrightarrow$

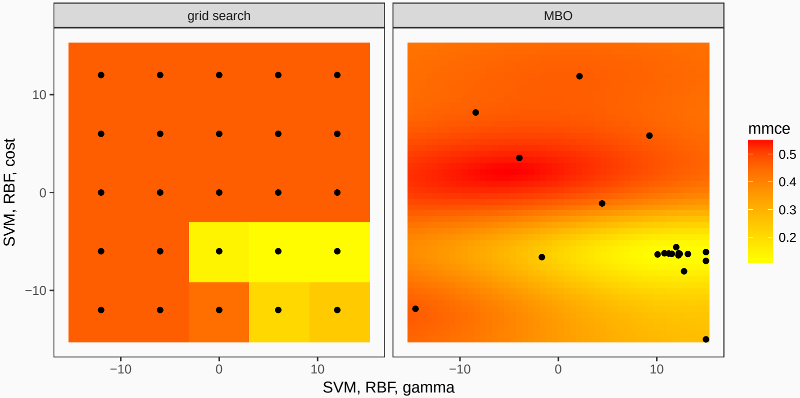
Motivation: Hyperparameter Tuning



- Because of budget restrictions grid might even be smaller!
- Unpromising area quite big!
- Lots of costly evaluations!

With `mlrMBO` it's not hard to do it better! $\hookrightarrow$

Motivation: Grid Search vs. MBO

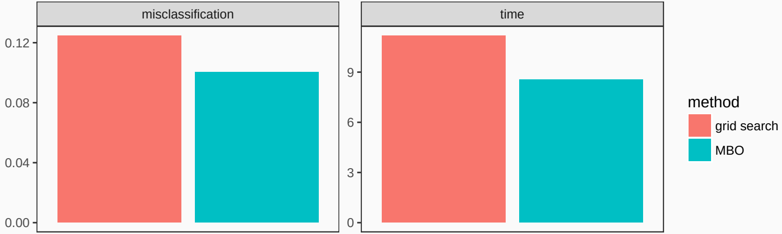


Motivation: Hyperparameter Tuning

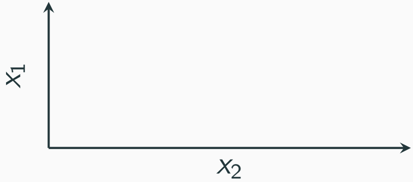
Compare results:

```
## Tune result:
## Op. pars: cost=4.1e+03; gamma=0.0156
## mmce.test.mean=0.1247619
## [1] 11.186
```

```
## Tune result:
## Op. pars: cost=4.8e+03; gamma=0.0135
## mmce.test.mean=0.1004762
## [1] 8.551
```

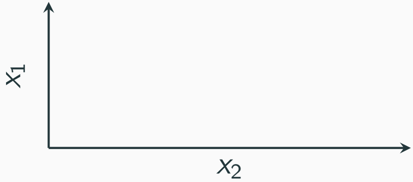


MBO: Illustrative Example



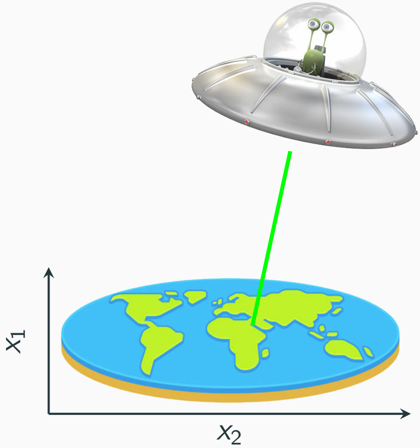
MBO: Illustrative Example

Problem: Alien is looking for the highest point on earth.



MBO: Illustrative Example

Problem: Alien is looking for the highest point on earth.



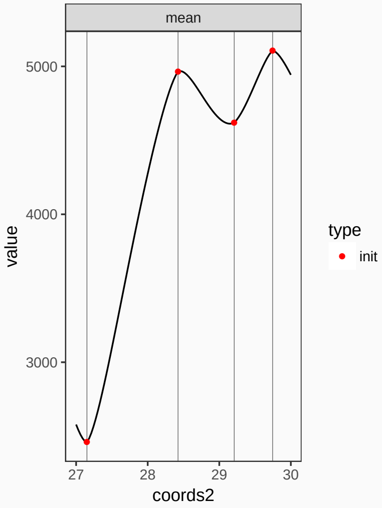
- Height can only be determined by complex ⌛ laser measurement.
- Laser can be set to (x_1, x_2) coordinate and returns the height (y) after some time.
- That's all our alien sees.

Illustrative Example

For simplification: Our alien got a hot tip to look at $x_1 = 86.92$ and $x_2 \in [27, 30]$.

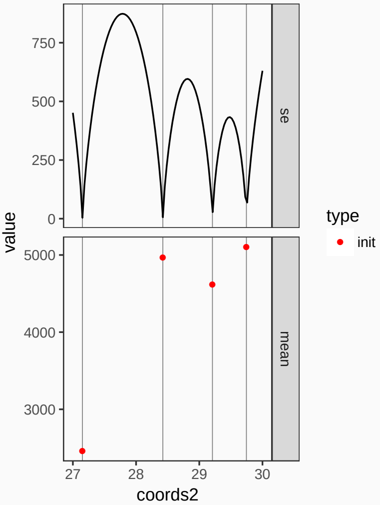
- No way to get information about the earth's surface except using the laser.
- Solution: Start with 4 "random" points.
  (usually LHS Sample)

Illustrative Example



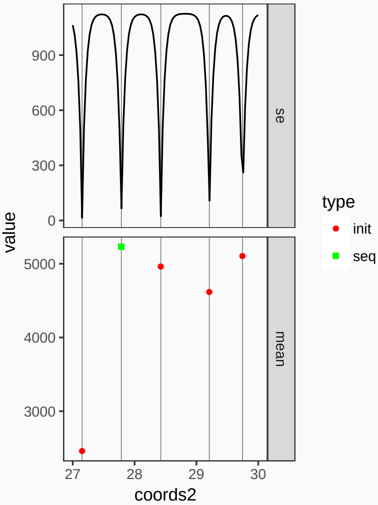
- Use regression methods (e.g. Kriging) to get prediction for unknown x_2.
- Prediction of $\hat{\mu}(x)$ does not help, as optimum apparently already known.

Illustrative Example



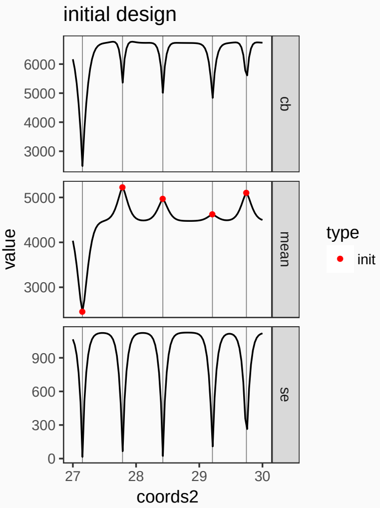
- Use regression methods (e.g. Kriging) to get prediction for unknown x_2.
- Prediction of $\hat{\mu}(x)$ does not help, as optimum apparently already known.
- We need to explore: Use estimate $\hat{s}(x)$ to find uncertain regions.

Illustrative Example



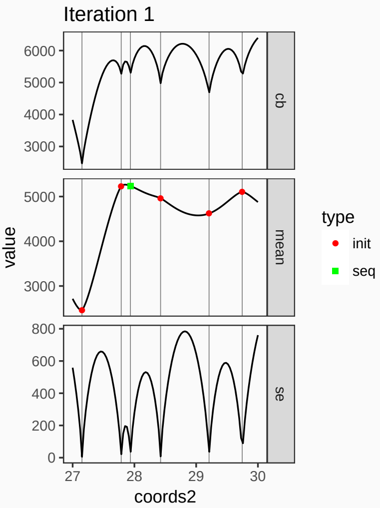
- Use regression methods (e.g. Kriging) to get prediction for unknown x_2.
- Prediction of $\hat{\mu}(x)$ does not help, as optimum apparently already known.
- We need to explore: Use estimate $\hat{s}(x)$ to find uncertain regions.
- "Bad" areas with high uncertainty uninteresting.

Illustrative Example



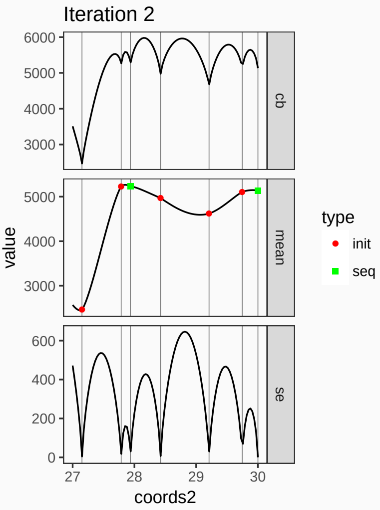
- Use regression methods (e.g. Kriging) to get prediction for unknown x_2.
- Prediction of $\hat{\mu}(x)$ does not help, as optimum apparently already known.
- We need to explore: Use estimate $\hat{s}(x)$ to find uncertain regions.
- "Bad" areas with high uncertainty uninteresting.
- Combine mean prediction and uncertainty using Infill Criterion: $CB(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}) + \lambda \cdot \hat{s}(\boldsymbol{x})$.

Illustrative Example



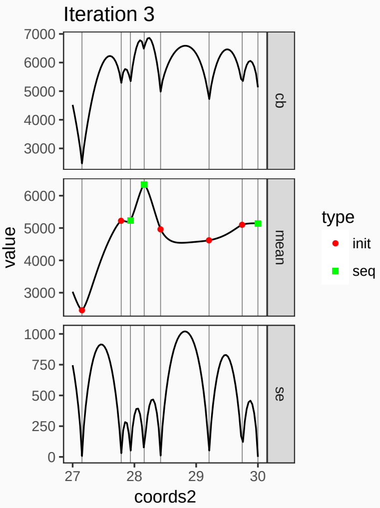
- Use regression methods (e.g. Kriging) to get prediction for unknown x_2.
- Prediction of $\hat{\mu}(x)$ does not help, as optimum apparently already known.
- We need to explore: Use estimate $\hat{s}(x)$ to find uncertain regions.
- "Bad" areas with high uncertainty uninteresting.
- Combine mean prediction and uncertainty using Infill Criterion: $CB(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}) + \lambda \cdot \hat{s}(\boldsymbol{x})$.

Illustrative Example



- Use regression methods (e.g. Kriging) to get prediction for unknown x_2.
- Prediction of $\hat{\mu}(x)$ does not help, as optimum apparently already known.
- We need to explore: Use estimate $\hat{s}(x)$ to find uncertain regions.
- "Bad" areas with high uncertainty uninteresting.
- Combine mean prediction and uncertainty using Infill Criterion: $CB(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}) + \lambda \cdot \hat{s}(\boldsymbol{x})$.

Illustrative Example



- Use regression methods (e.g. Kriging) to get prediction for unknown x_2.
- Prediction of $\hat{\mu}(x)$ does not help, as optimum apparently already known.
- We need to explore: Use estimate $\hat{s}(x)$ to find uncertain regions.
- "Bad" areas with high uncertainty uninteresting.
- Combine mean prediction and uncertainty using Infill Criterion: $CB(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}) + \lambda \cdot \hat{s}(\boldsymbol{x})$.

mlrMBO: Introduction

The Package: `mlrMBO`

Insights

- 5+ years old
- 11 contributers
- 6000+ lines of tested code
- ~ 1000 monthly r-studio `CRAN` downloads
- Base for multiple papers

- Documentation: `https://mlr-org.github.io/mlrMBO/`
- Bug + Issue Tacker: `https://github.com/mlr-org/mlrMBO/issues`

Get Started

Using predefined benchmark function from `smoof` Package to start with all defaults:

```
library(mlrMBO)
ctrl = makeMBOControl()
fun = makeBraninFunction()
res = mbo(fun, control = ctrl)
res$x
```

```
## $x
## [1] 9.486302 2.368736
```

```
res$y
```

```
## [1] 0.4412247
```

Termination

Control budget of an MBO-Run

- Iterations after initial design
- Maximum evaluations of objective function including initial design
- Maximum total time budget
- Maximum net execution runtime of objective function
- Threshold for target function value

```
ctrl = makeMBOControl()

ctrl = setMBOControlTermination(ctrl,
  iters = 20, max.evals = 10, time.budget = 4,
  exec.time.budget = 2, target.fun.value = 0.01)
res = mbo(fun, control = ctrl)

res$final.state


## [1] "term.feval"
```

First met condition determines termination.
Custom termination criteria can be implemented!

Objective Functions

Objective functions are wrapped in `smoof` functions. They contain:

- name,
- the function,
- definition of the domain (search space),
- optimization direction
- and further meta information ...

```
fun = makeSingleObjectiveFunction(
  id = "simple.example",
  fn = function(x) x[1]^2 * sin(x[2]),
  par.set = makeNumericParamSet("x", len = 2, lower = -5, upper = 5),
  minimize = TRUE
)
```

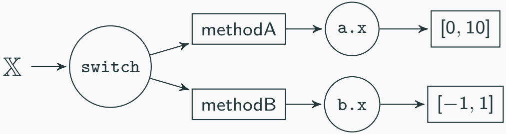
`smoof` and `ParamHelpers` Package

Wrap external functions with
`smoof::makeSingleObjectiveFunction()`.
The search space is always described in a *Parameter Set*.

```
fun = makeSingleObjectiveFunction(
  id = "example",
  fn = function(x)
    complicatedFunction(x$a, x$b, c = 10, conf = list(d = x$d, e = x$e, f = x$f)),
  par.set = makeParamSet(
    makeDiscreteParam("a", values = c("foo", "bar")),
    makeIntegerParam("b", lower = 0, upper = 10),
    makeNumericParam("d", lower = -5, upper = 5, trafo = function(x) 2^x),
    makeLogicalParam("e"),
    makeDiscreteParam("f", list("sin" = sin, "cos" = cos))
  ),
  minimize = TRUE, has.simple.signature = FALSE
)
x = sampleValue(getParamSet(fun), trafo = TRUE)
fun(x)


## [1] 11
```

The *Parameter Set* can even contain complex dependencies:




```r
ps = makeParamSet(
  makeDiscreteParam("switch", values = c("methodA", "methodB")),
  makeNumericParam("a.x", 0, 10, requires = quote(switch == "methodA")),
  makeNumericParam("b.x", -1, 1, requires = quote(switch == "methodB"))
)
```

`smoof`: Arguments

- Single objective: `makeSingleObjectiveFunction()`
  - non deterministic: `noisy = TRUE`
  - arguments as list: `has.simple.signature = FALSE`
  - maximize: `minimize = FALSE`
- Multi objective: `makeMultiObjectiveFunction()`
  - number of objectives: `n.objectives`
  - non deterministic: `noisy = TRUE`
  - arguments as list: `has.simple.signature = FALSE`
  - maximize: e.g.: `minimize = c(FALSE, FALSE)`

Initial Design

```
mbo(..., design = des, ...)
```

Default:

- MBO draws LHS-Sample with $4 * d$ points.
- MBO first evaluates initial design.

Options:

- Pass design of x-values (one per row)
  - e.g. `ParamHelpers::generateDesign()`
- Pass design of x and y-values
  - Saves computation time if results are already known.

Use-cases for manual designs:

- Specific values known that perform well.
- Results of previous evaluations.

mlrMBO for Hyperparameter Optimization

mlr: Define Objective Function

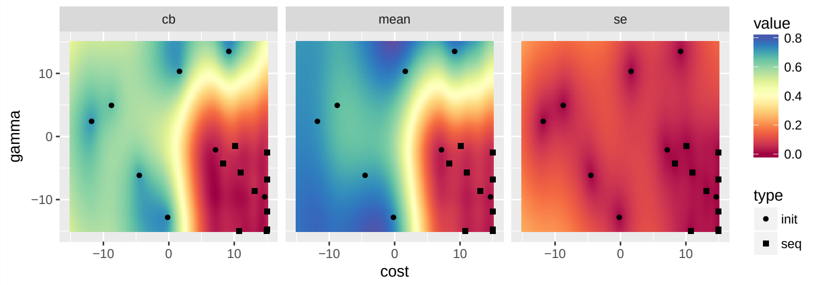
Define objective function as the performance measured by a resampling done with mlr[1]:

```
par.set = makeParamSet(
  makeNumericParam("cost", -15, 15, trafo = function(x) 2^x),
  makeNumericParam("gamma", -15, 15, trafo = function(x) 2^x)
)
svm = makeSingleObjectiveFunction(name = "svm.tuning",
  fn = function(x) {
    lrn = makeLearner("classif.svm", par.vals = x)
    resample(lrn, iris.task, cv3, show.info = FALSE)$aggr
  },
  par.set = par.set, noisy = TRUE,
  has.simple.signature = FALSE, minimize = TRUE
)
ctrl = makeMBOControl()
ctrl = setMBOControlTermination(ctrl, iters = 10)
res = mbo(svm, control = ctrl, show.info = FALSE)
```

[1] Bischl, Lang, et al. "mlr: Machine Learning in R". 2016.

Define Objective Function

```
plot(res$final.opt.state)
```




```
kable(tail(as.data.frame(res$opt.path)[,c("cost", "gamma", "y", "dob",
  "exec.time", "train.time")], 4))
```

	cost	gamma	y	dob	exec.time	train.time
15	14.99993	-14.77408	0.0333333	7	0.293	0.350
16	14.99988	-11.92746	0.0400000	8	0.305	0.319
17	14.98988	-14.99882	0.0333333	9	0.344	0.408
18	10.77922	-14.99781	0.0400000	10	0.326	0.216

Use mlr tuning interface

```
ctrl = makeMBOControl()
ctrl = setMBOControlTermination(ctrl, iters = 10)
tune.ctrl = makeTuneControlMBO(mbo.control = ctrl)
res = tuneParams(makeLearner("classif.svm"), iris.task, cv5,
  par.set = par.set, control = tune.ctrl, show.info = FALSE)
res
```

```
## Tune result:
## Op. pars: cost=7.49e+03; gamma=4.95e-05
## mmce.test.mean=0.0266667
```

```
kable(tail(as.data.frame(res$opt.path), 4))
```

	cost	gamma	mmce.test.mean	dob	eol	error.message	exec.time
15	14.99809	-14.998977	0.0333333	15	NA	NA	0.173
16	12.87146	-14.301882	0.0266667	16	NA	NA	0.200
17	13.84028	-4.793858	0.0466667	17	NA	NA	0.173
18	11.56522	-1.016864	0.0600000	18	NA	NA	0.105

Advanced Settings for MBO

Surrogate Model

Default:

- Kriging (`mlr`: `"regr.km"`) for numerical search spaces.
- Random Forest (`mlr`: `"regr.randomForest"`) otherwise.

Options:

- All regression learners integrated in `mlr`.
- `pred.type = "se"` needed for *infill criteria*.
- Wrap learners with `mlr` wrappers for additional functionality.

Notes:

- `"regr.km"` can crash sometimes
- `"regr.GPfit"` more stable

Infill Criteria

Possible infill criteria:

- Mean Response: `crit.mr` (no exploration)
- Uncertainty: `crit.se` (no exploitation)
- Confidence Bound: `crit.cb`, `makeMBOInfillCritCB(lambda = 3)`
- **Expected Improvement**: `crit.ei`
- Noisy objective function
  - Expected Quantile Improvement: `crit.eqi`
  - Augmented Expected Improvement: `crit.aei`

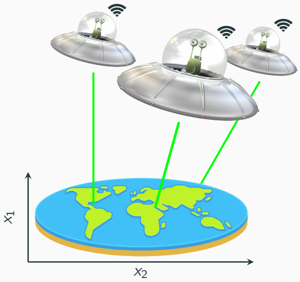
Advanced Hints

Frequently requested topics:

- Optimization Path: `as.data.frame(res$opt.path)`
- Use MBO to optimize a Algorithm via CLI: `mlr-org.github.io/mlrMBO/articles/supplementary/mlrmbo_and_the_command_line.html`
- Investigate surrogate model: `makeMBOControl(store.model.at = c(1,5,10), ...)`
- Continue if surrogate model crashes: `makeMBOControl(on.surrogate.error = "warn", ...)`
- Continue if objective function returns `NA`: `makeMBOControl(impute.y.fun = function(x, y, opt.path) 0, ...)`
- Visualization: `runExampleRun()`, `plotExampleRun()` or ...
- Human in the Loop `https://mlr-org.github.io/mlrMBO/articles/supplementary/human_in_the_loop_MBO.html`

Multi-point Proposals and Parallelization

Scenarios?



Objective function . . .

. . . can be parallelized?

$\Rightarrow$ parallelize objective function.

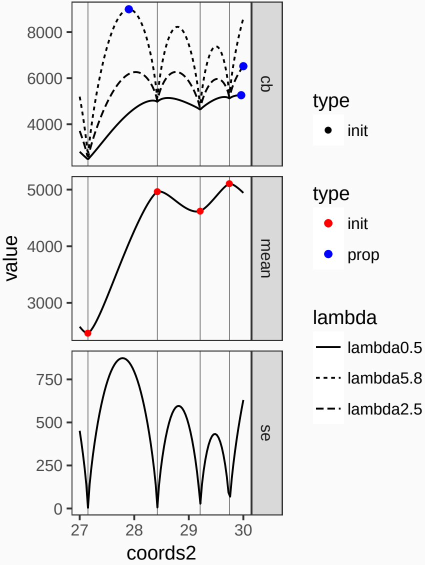
. . . can not be further parallelized / still available resources.

$\Rightarrow$ use multi-point proposals!

Proposition methods:

- Constant Liar: Iterative, suggests point, adds preliminary fictitious result into the design. *(costly)*
- qCB: Vary uncertainty weights. *(cheaper)*
- . . .

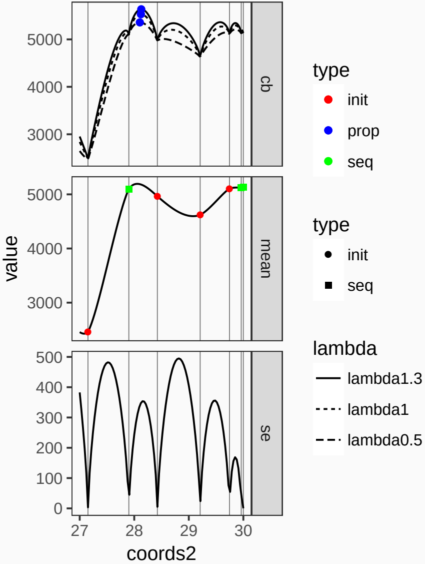
Example: qCB



$$CB(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}) + \lambda \cdot \hat{s}(\boldsymbol{x})$$

- CB with small λ: search close to known optimum: *exploitation*.
- CB with high λ: explore unevaluated areas: *exploration*.
- Problem: Points can be close to each other.
- Solution: Use *Constant Liar*.

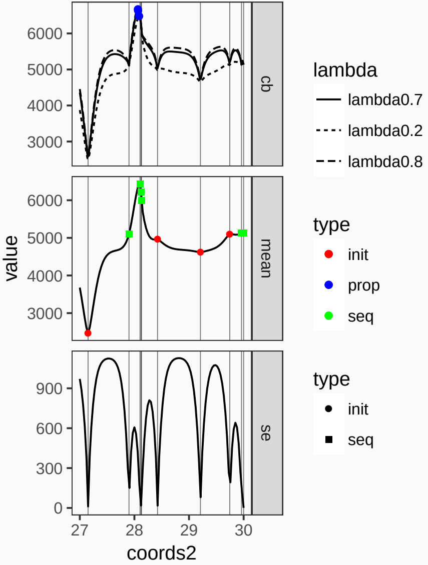
Example: qCB



$$CB(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}) + \lambda \cdot \hat{s}(\boldsymbol{x})$$

- CB with small λ: search close to known optimum: *exploitation*.
- CB with high λ: explore unevaluated areas: *exploration*.
- Problem: Points can be close to each other.
- Solution: Use *Constant Liar*.

Example: qCB



$$CB(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}) + \lambda \cdot \hat{s}(\boldsymbol{x})$$

- CB with small λ: search close to known optimum: *exploitation*.
- CB with high λ: explore unevaluated areas: *exploration*.
- Problem: Points can be close to each other.
- Solution: Use *Constant Liar*.

Example: Parallelization

Use *Expected Improvement* as infill criterion and the *constant liar* method to generate multiple proposals:

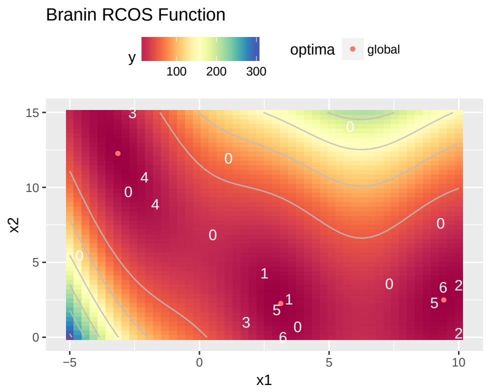
```
set.seed(1)
obj.fun = makeBraninFunction()
ctrl = makeMBOControl(propose.points = 2)
ctrl = setMBOControlInfill(ctrl, crit = crit.ei)
ctrl = setMBOControlMultiPoint(ctrl, method = "cl", cl.lie = min)
ctrl = setMBOControlTermination(ctrl, iters = 6)
library(parallelMap)
parallelStartMulticore(cpus = 2, level = "mlrMBO.feval")
res = mbo(obj.fun, control = ctrl, show.info = FALSE)
parallelStop()
res
```

```
## Recommended parameters:
## x=2.98,1.84
## Objective: y = 0.839
##
## Optimization path
## 8 + 12 entries in total, displaying last 10 (or less):
##          x1           x2          y dob eol error.message
## 11  9.999782 2.710021e-01  9.4043915   2  NA          <NA>
## 12  9.999953 3.486280e+00  2.1765417   2  NA          <NA>
## 13  1.802756 1.006915e+00 14.2632797   3  NA          <NA>
```

Example: Parallelization

Use the points in the *Optimization Path* and plot them over the true response surface of the objective function:

```
autoplot(obj.fun, render.levels = TRUE, show.optimum = TRUE) +
geom_text(data = as.data.frame(res$opt.path), mapping = aes(label = dob), color = "white")
```



Parallelization in combination with `mlr`

Parallelize resampling:

```
parallelStartMulticore(3, level = "mlr.resample")
res = tuneParams(makeLearner("classif.svm"), iris.task, cv3,
  par.set = par.set, control = tune.ctrl)
parallelStop()
```

Parallelize multiple evaluations with multi-point proposal:

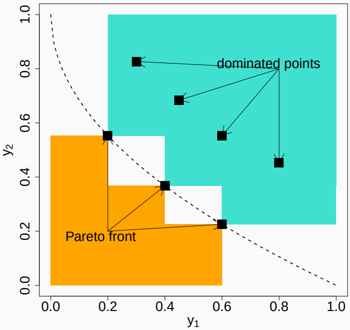
```
ctrl = setMBOControlMultiPoint(ctrl, method = "cl", cl.lie = min)
tune.ctrl = makeTuneControlMBO(mbo.control = ctrl)
parallelStartMulticore(2, level = "mlrMBO.feval")
res = tuneParams(makeLearner("classif.svm"), iris.task, holdout,
  par.set = par.set, control = tune.ctrl)
parallelStop()
```

Multi-objective optimization

❗ Goal: Optimize multiple objectives:

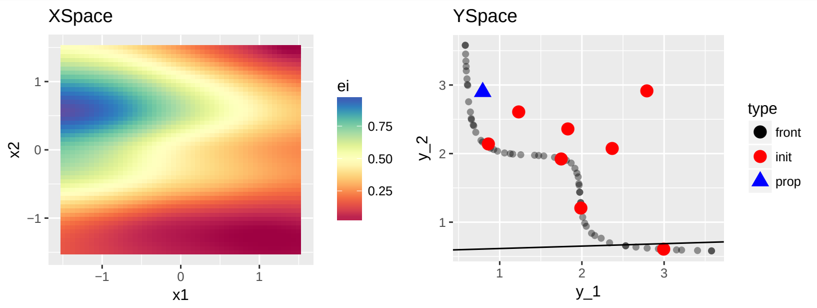
e.g. maximize *True positive rate* and

- minimize *False positive rate* at the same time.



ParEGO

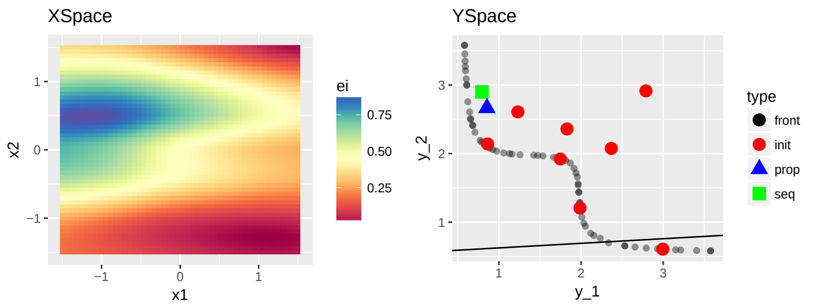
ParEGO[2] with one proposal per iteration. Line — indicates weight vector $\vec{w}$ for scalarization $\tilde{y} = \vec{w}\boldsymbol{y}$. EI is calculated for $\hat{\tilde{y}}$.



2 Knowles. "ParEGO: A Hybrid Algorithm with on-Line Landscape Approximation for Expensive Multiobjective Optimization Problems". 2006.

ParEGO

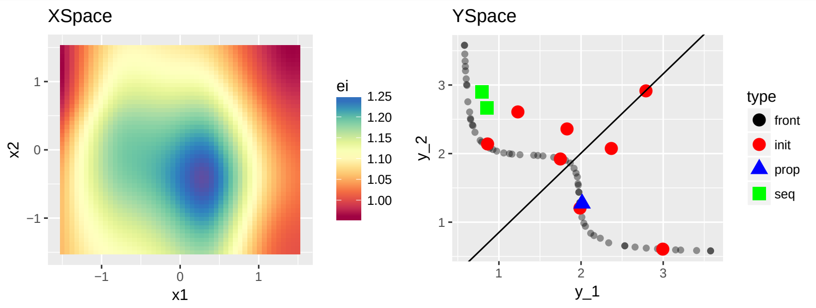
ParEGO[2] with one proposal per iteration. Line — indicates weight vector $\vec{w}$ for scalarization $\tilde{y} = \vec{w}\boldsymbol{y}$. EI is calculated for $\hat{\tilde{y}}$.



2 Knowles. "ParEGO: A Hybrid Algorithm with on-Line Landscape Approximation for Expensive Multiobjective Optimization Problems". 2006.

ParEGO

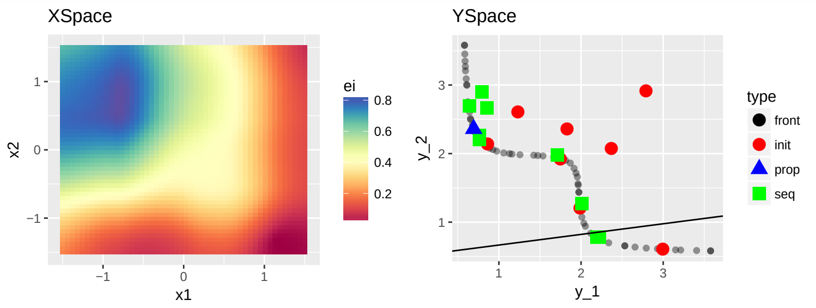
ParEGO[2] with one proposal per iteration. Line — indicates weight vector $\vec{w}$ for scalarization $\tilde{y} = \vec{w}\boldsymbol{y}$. *EI* is calculated for $\hat{\tilde{y}}$.



2 Knowles. "ParEGO: A Hybrid Algorithm with on-Line Landscape Approximation for Expensive Multiobjective Optimization Problems". 2006.

ParEGO

ParEGO[2] with one proposal per iteration. Line — indicates weight vector $\vec{w}$ for scalarization $\tilde{y} = \vec{w}\boldsymbol{y}$. *EI* is calculated for $\hat{\tilde{y}}$.



2 Knowles. "ParEGO: A Hybrid Algorithm with on-Line Landscape Approximation for Expensive Multiobjective Optimization Problems". 2006.

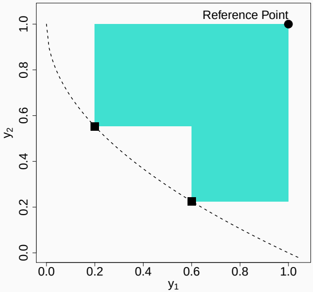
ParEGO

```r
set.seed(1)
fun = makeDentFunction()
ctrl = makeMBOControl(n.objectives = 2)
ctrl = setMBOControlInfill(ctrl, crit = crit.ei)
ctrl = setMBOControlMultiObj(ctrl, method = "parego")
res = mbo(fun, control = ctrl)
res
```

```
##           y_1       y_2
## 1  0.9906996 2.1556803
## 2  2.4300056 0.6807635
## 6  0.8296992 2.3186675
## 10 3.5808754 0.5812537
## 14 1.7734341 1.9075915
## 15 0.7184362 3.0187745
## 16 2.0689133 0.9307282
## 18 1.5626290 1.9629775
## Optimization path
## 8 + 10 entries in total, displaying last 10 (or less):
##              x1          x2       y_1       y_2 dob eol
## 9  -0.43125323  1.49782410 0.8734869 2.8025643   1  NA
## 10  1.49995759 -1.49966412 3.5808754 0.5812537   2  NA
## 11 -1.01627755 -0.87934065 2.3420145 2.4789514   3  NA
## 12  0.09084243  0.37442531 1.7137141 1.9972969   4  NA
## 13  1.41830629 -0.40148093 2.6921849 0.8723977   5  NA
```

SMS-EGO[3]

- Single-objective optimization of aggregating infill criterion: Calculate contribution of an "optimistic estimate" ($LCB(\boldsymbol{x}) = \hat{\boldsymbol{y}} - \lambda \cdot \hat{\mathbf{s}}^2$) to the current Pareto front approximation $\boldsymbol{\Lambda}$.

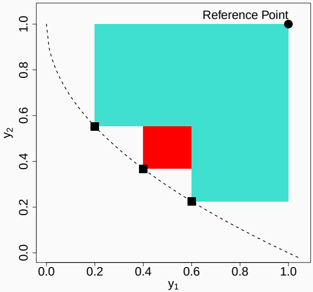


- Calculate LCB for each objective
- Measure contribution with regard to the hypervolume indicator.
- Propose point with highest estimated hypervolume contribution $\mathcal{S}(LCB(\boldsymbol{x}) \cap \boldsymbol{\Lambda}) - \mathcal{S}(\boldsymbol{\Lambda})$.

[3] Ponweiser et al. "Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted $\mathcal{S}$-Metric Selection". 2008.

SMS-EGO[3]

- Single-objective optimization of aggregating infill criterion: Calculate contribution of an "optimistic estimate" ($LCB(\boldsymbol{x}) = \hat{\boldsymbol{y}} - \lambda \cdot \hat{\mathbf{s}}^2$) to the current Pareto front approximation $\boldsymbol{\Lambda}$.

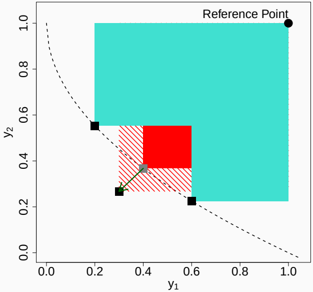


- Calculate LCB for each objective
- Measure contribution with regard to the hypervolume indicator.
- Propose point with highest estimated hypervolume contribution $\mathcal{S}(LCB(\boldsymbol{x}) \cap \boldsymbol{\Lambda}) - \mathcal{S}(\boldsymbol{\Lambda})$.

3 Ponweiser et al. "Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted $\mathcal{S}$-Metric Selection". 2008.

SMS-EGO[3]

- Single-objective optimization of aggregating infill criterion: Calculate contribution of an "optimistic estimate" ($LCB(\boldsymbol{x}) = \hat{\boldsymbol{y}} - \lambda \cdot \hat{\mathbf{s}}^2$) to the current Pareto front approximation $\boldsymbol{\Lambda}$.



- Calculate LCB for each objective
- Measure contribution with regard to the hypervolume indicator.
- Propose point with highest estimated hypervolume contribution $\mathcal{S}(LCB(\boldsymbol{x}) \cap \boldsymbol{\Lambda}) - \mathcal{S}(\boldsymbol{\Lambda})$.

[3] Ponweiser et al. "Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted $\mathcal{S}$-Metric Selection". 2008.

SMS-EGO

```
set.seed(1)
fun = makeDentFunction()
ctrl = makeMBOControl(n.objectives = 2)
ctrl = setMBOControlInfill(ctrl, crit = crit.dib1)
ctrl = setMBOControlMultiObj(ctrl, method = "dib")
res = mbo(fun, control = ctrl)
res
```

```
##           y_1       y_2
## 2  2.4300056 0.6807635
## 4  1.7385428 2.0070887
## 9  3.5806607 0.5812595
## 10 0.6551836 3.1617681
## 12 0.7176490 3.0202811
## 14 2.0430301 1.3404486
## 15 0.7560033 2.2157578
## 17 0.8392660 2.1187746
## 18 2.2612374 0.7689810
## Optimization path
## 8 + 10 entries in total, displaying last 10 (or less):
##             x1         x2       y_1       y_2 dob eol
## 9   1.4995956 -1.4998056 3.5806607 0.5812595   1  NA
## 10 -1.4999711  1.0066134 0.6551836 3.1617681   2  NA
## 11  0.4808214  0.5452640 2.0316819 2.0961246   3  NA
## 12 -0.8027083  1.4999239 0.7176490 3.0202811   4  NA
```

Multi-objective optimization with mlr

Without mbo.control it defaults to DIB and the budget becomes max.eval.

```
set.seed(1)
par.set = makeParamSet(
  makeNumericParam("cost", -15, 15, trafo = function(x) 2^x),
  makeNumericParam("gamma", -15, 15, trafo = function(x) 2^x)
)
ctrl = makeTuneMultiCritControlMBO(n.objectives = 2L, budget = 20)
res = tuneParamsMultiCrit("classif.svm", sonar.task, cv3, par.set = par.set,
    measures = list(tpr, fpr), control = ctrl)
res$y
```

```
##    tpr.test.mean fpr.test.mean
## 3      1.0000000     0.8247312
## 19     0.9304993     0.1715054
## 20     0.9304993     0.1715054
```

Complex Example

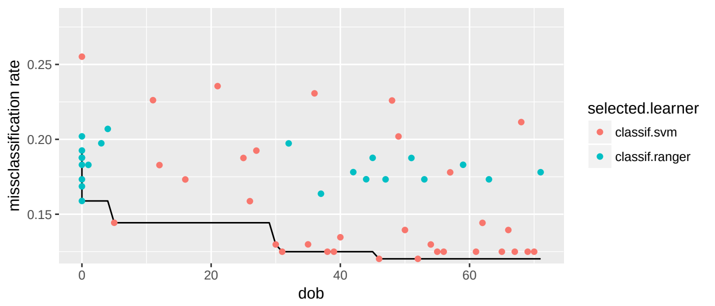
Complex Example: mlr Model Multiplexer

```r
library(mlrCPO); library(dplyr); library(mlrMBO)
lrn = c("classif.svm", "classif.ranger") %>% makeLearners() %>%
  makeModelMultiplexer()
ps = makeModelMultiplexerParamSet(lrn,
  classif.svm = makeParamSet(
    makeNumericParam("cost", -15, 15, trafo = function(x) 2^x),
    makeNumericParam("gamma", -15, 15, trafo = function(x) 2^x)),
  classif.ranger = makeParamSet(
    makeIntegerParam("mtry", lower = 1L, upper = 60L)
  )
)
sur.lrn = cpoImputeAll(id = "imp", classes = list(numeric = imputeMax(2))) %>>%
  cpoDummyEncode(id = "dum") %>>% makeLearner("regr.km", predict.type = "se")
ctrl = makeMBOControl() %>% setMBOControlTermination(time.budget = 60) %>%
  setMBOControlInfill(crit.ei) %>% makeTuneControlMBO(mbo.control = .,
  learner = sur.lrn)
res = tuneParams(lrn, sonar.task, cv3, control = ctrl, par.set = ps)
str(res$x)


## List of 3
##  $ selected.learner : chr "classif.svm"
##  $ classif.svm.cost : num 41.5
##  $ classif.svm.gamma: num 0.0177
```

Complex Example: `mlr` Model Multiplexer

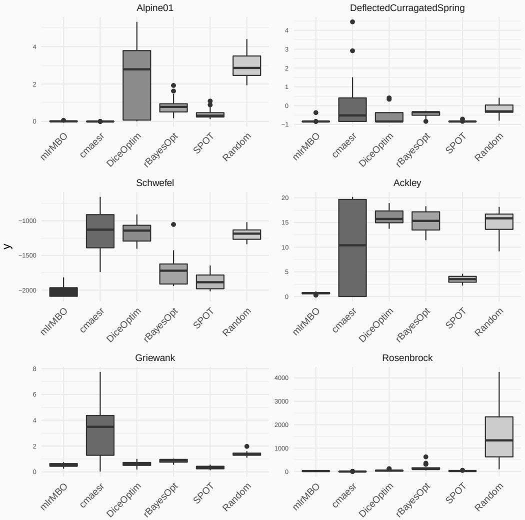
```
opdf = as.data.frame(res$mbo.result$opt.path)
library(ggplot2)
g = ggplot(opdf, aes(x = dob, y = cummin(y)))
g = g + geom_line() + geom_point(aes(color = selected.learner, y = y))
g + coord_cartesian(ylim = c(0.125,0.28)) + ylab("missclassification rate")
```



Conclusion

Performance

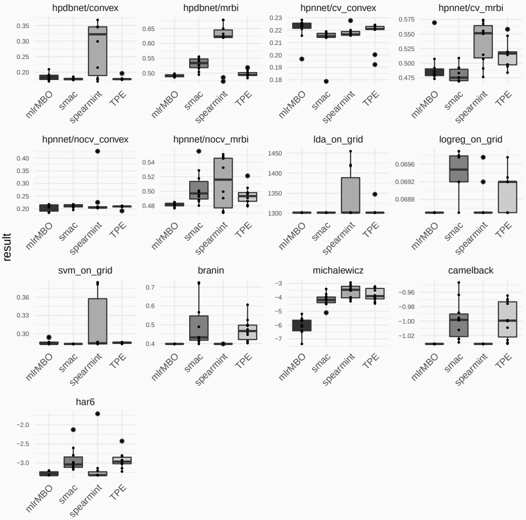
Comparison of different Black-Box optimizers available in R[4]:



[4] Bischl, Richter, et al. "mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions". 2017.

Performance

mlrMBO vs. other Black-Box optimizers on HPOlib benchmark[5]:



5 Bischl, Richter, et al. "mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions". 2017.

Conclusion

Key features

- Highly customizable expensive Black-Box optimization
- Integrated parallelization
- Multi-objective optimization
- Seamless `mlr` integration

Resources

Help: `https://mlr-org.github.io/mlrMBO` ⇒ Topics

Bug + Issue Tracker:
`https://github.com/mlr-org/mlrMBO/issues`

Slack Chanel #mlrMBO: `https://mlr-org.slack.com/`

References

Bischl, Bernd, Michel Lang, et al. (2016). “mlr: Machine Learning in R”. In: *Journal of Machine Learning Research* 17.170, pp. 1–5. URL: http://jmlr.org/papers/v17/15-066.html.

Bischl, Bernd, Jakob Richter, et al. (2017). “mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions”. In: *arXiv:1703.03373 [stat]*. arXiv: 1703.03373 [stat].

Knowles, J. (Feb. 2006). “ParEGO: A Hybrid Algorithm with on-Line Landscape Approximation for Expensive Multiobjective Optimization Problems”. In: *IEEE Transactions on Evolutionary Computation* 10.1, pp. 50–66. ISSN: 1089-778X. DOI: 10.1109/TEVC.2005.851274.

Ponweiser, Wolfgang et al. (Sept. 13, 2008). "Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted $\mathcal{S}$-Metric Selection". In: *Parallel Problem Solving from Nature – PPSN X*. International Conference on Parallel Problem Solving from Nature. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 784–794. ISBN: 978-3-540-87699-1 978-3-540-87700-4. DOI: 10.1007/978-3-540-87700-4_78. URL: https://link.springer.com/chapter/10.1007/978-3-540-87700-4_78 (visited on 10/26/2017).