Ludwig-Maximilians-Universität München

Department of Statistics

Bachelor's Thesis submitted in partial fulfillment of the requirements for the degree
Bachelor of Science

Visual Diagnostics for Bayesian Optimization

Reviewer:	Prof. Dr. Bernd Bischl Chair of Statistical Learning and Data Science Department of Statistics Ludwig-Maximilians-Universität München
Advisor:	Julia Moosbauer
Study program:	Statistics (B. Sc.)
Composed by:	Philipp Scheller
Date of submission:	22nd July 2020

I assure that the single-handed composition of this bachelor's thesis is only supported by the declared resources. It was not used in the same or in a similar version to achieve an academic grading or is being published elsewhere.

Munich, 22nd July 2020

[illegible]

Philipp Scheller

Abstract

Nowadays, Bayesian Optimization (BO) is applied in various disciplines of work. While the procedure of BO is often intransparent and the user-specifications are unchallenged, it limits the understanding and the potential for improvement of such models. Therefore, as part of this work, we present the R package *VisBayesOpt* which helps to overcome these difficulties. The work contributes in the following ways: First, we introduce Sequential Model-Based Optimization (SMBO) in a formal way. Second, we review existing packages in the programming languages R and Python to provide an overview on the status quo of visualization tools for SMBO. Third, we introduce the package *VisBayesOpt* and its user friendly Shiny application which facilitates the analysis of SMBO runs. Fourth, we conduct an exemplary analysis of a SMBO problem in the context of Machine Learning (ML), that we analyze by using the visualizations of *VisBayesOpt*. We conclude the work with an outlook on the usage of *VisBayesOpt* in the future and discuss further improvements of the package.

Zusammenfassung

Heutzutage findet die BO Anwendung in einer Vielzahl von Arbeitsbereichen. Während das Verfahren der BO oft intransparent bleibt und die Nutzerspezifikationen wenig hinterfragt werden, führt dies zu einem eingeschränkten Verständnis und limitiert das Verbesserungspotential solcher Modelle. Daher stellen wir im Rahmen dieser Arbeit das R Paket *VisBayesOpt* vor, welches bei diesen Schwierigkeiten unterstützt. Der Beitrag der Arbeit is wie folgt: Zunächst führen wir die SMBO auf formale Weise ein. Zweitens überprüfen wir bestehende Pakete in den Programmiersprachen R und Python, um einen Überblick über den gegenwärtigen Status von Visualisierungstools für SMBO darzulegen. Drittens stellen wir das Paket *VisBayesOpt* und seine benutzerfreundliche Shiny Applikation vor, welche die Analyse von SMBO-Läufen erleichtern soll. Viertens betrachten wir ein SMBO-Problem, aus dem Anwendungsbereich des maschinellen Lernens, welches wir mit Hilfe der Visualisierungen aus dem Paket *VisBayesOpt* analysieren. Wir runden die Arbeit mit einem Ausblick auf die zukünftige Verwendung von *VisBayesOpt* ab und diskutieren weitere Verbesserungsmöglichkeiten des Pakets.

Contents

List of Figures

List of Tables

List of Abbreviations

AUC	Area Under the Curve
BO	Bayesian Optimization
BBO	Black-box Optimization
EI	Expected Improvement
GP	Gaussian Process
iid	Independent and identically distributed
IML	Interpretable Machine Learning
LCB	Lower Confidence Bound
ML	Machine Learning
PDP	Partial Dependence Plot
PI	Probability of Improvement
RF	Random Forest
SE	Squared Exponential
SMBO	Sequential Model-Based Optimization
UI	User Interface
UML	Unified Modeling Language
w.r.t.	With respect to
XGBoost	eXtreme Gradient Boosting

List of Symbols

$\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, \ldots, n\} \in (\mathcal{X} \times \mathcal{Y})$	Set of n observations (design)
$\mathcal{D}_\star = \{(\mathbf{x}_\star^{(i)}, y_\star^{(i)}) \mid i = 1, \ldots, n_\star\}$	Test-set of $n_\star$ unobserved points
$\mathbb{E}(\cdot)$	Expectation function
$f : \mathcal{X} \to \mathcal{Y}$	(Unknown) objective function / Black-box function
$\hat{f}$	Surrogate model
$\mathbf{f}_\star$	GP posterior at test points, i.e. $\mathbf{f} = \big(f(\mathbf{x}_\star^{(1)}), \ldots, f(\mathbf{x}_\star^{(n_\star)})\big)$, where $f \sim \mathcal{GP}$
$\mathcal{GP}$	Gaussian process; $f \sim \mathcal{GP}\big(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\big)$ is a GP with mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$
$k(\mathbf{x}, \mathbf{x}')$	Covariance function of GP
$\mathbf{K} = \big(k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})\big)_{i,j=1\ldots,n}$	Prior covariance matrix of GP
$\mathbf{K}_\star = \big(k(\mathbf{x}_\star^{(i)}, \mathbf{x}_\star^{(j)})\big)_{i,j=1\ldots,n_\star}$	Posterior covariance matrix of GP
l	Characteristic length-scale of SE-kernel
$m(\mathbf{x})$	Mean function of GP
$\mathbf{m}_\star = \big(m(\mathbf{x}_\star^{(i)})\big)_{i=1,\ldots,n_\star}$	Posterior mean of GP
$\mathbb{P}_{X,Y}$	Joint probability distribution of $\mathcal{X} \times \mathcal{Y}$; $(\mathbf{X}, \mathbf{Y}) \overset{\text{iid}}{\sim} \mathbb{P}_{X,Y}$
$\hat{s}(\mathbf{x})$	Estimate of standard deviation of surrogate
S	Number of decision trees in RF
$T_i(X)$	Output provided by the i-th decision tree
$\mathbb{V}(\cdot)$	Variance function
$\mathcal{X}$	Search space of dimension p
$\mathbf{x} = (x_1, \ldots, x_p)^\top \in \mathcal{X}$	Observed search space vector
$\mathbf{X} = \big(x^{(1)}, \ldots, x^{(n)}\big)^\top$	Observed training input
$\mathbf{X}_\star = \big(x_\star^{(1)}, \ldots, x_\star^{(n_\star)}\big)^\top$	Un-observed test input
$\mathbf{x}^\star$	Set of search space values which minimizes f
$y \in \mathcal{Y}$	Observed (noisy) output vector $\big(y = f(\mathbf{x}) + \epsilon, \epsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)\big)$
$\mathbf{y}$	GP prior for noisy observations, i.e. $\mathbf{y} = \big(y^{(1)}, \ldots, y^{(n)}\big)$, where $y \sim f(\mathbf{x}) + \epsilon, \epsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)\big)$ and $f \sim \mathcal{GP}$
$\mathcal{Y}$	Output space

$\Gamma(\cdot)$	Gamma function
δ_{ij}	Kronecker delta; 1 iff $i = j$, else 0
λ	Parameter to control exploration/exploitation of infill criterion
$\hat{\mu}(\mathbf{x})$	Mean estimate of surrogate model
$\phi(\cdot)$	Probability distribution function of normal distribution
$\Phi(\cdot)$	Cumulative distribution function of normal distribution
$\|\| \cdot \|\|$	Euclidean norm

1 The Importance of Black-box Optimization in Various Disciplines of Work

In recent years the world strives more for efficiency than ever before. Production and research use resources in the most efficient way, while tasks as such gain in complexity. Laboratory experiments for example need to consider various input parameters which all together have an impact on the measured output.[3][p. 5] The input and output is linked by a function which is often unknown. In a world without constraints, where time does not matter, resources are unlimited, and evaluations are cheap, we would try to observe all possible input-output combinations and choose the set of inputs that lead to the best result. Since most problems increase in complexity, trying all combinations is mostly not possible. Besides that the evaluation of many laboratory experiments nowadays is highly complex leading to high consumption of time and resources, which make each combination of the input/output-relation expensive to evaluate. The absence of a functional relation leads to the fact that traditional optimization, like derivative based approaches, cannot be applied to such kind of problems.

Black-Box Optimization (BBO) handles such problems where (i) the functional form between input and output is unknown and (ii) the influence of combinations of inputs on the output is expensive to evaluate and is usually free of any assumptions on continuity, differentiability, or smoothness.[3][p. 6] One of the most prominent approaches of BBO is the field of BO where optimization levers the Bayes Theorem by incorporating the evidence gathered during the optimization process. This knowledge is incorporated in the response surface or so called surrogate.[2][preface] SMBO, a prominent algorithm to perform BO, iterates between fitting models and using them to make choices about which configuration to evaluate.[11][p. 2] SMBO is steered by an acquisition function that balances the trade-off between exploration (sample next point where uncertainty is high) and exploitation (sample next point where function is expected to be minimized).[12][p. 455]

The demonstrated performance of SMBO, compared to other optimization methods, in many areas of expertise ([4], [9], [6]) makes SMBO a technique applied by people from various disciplines like physicist, mechanical engineers, seismologists, statisticians and many others. Several package implementations of BO exist in common statistical programming languages like R or Pyhton. Applying methods from these packages is easy and does not necessarily require in depth knowledge on statistics and the field of BO. Fist, this might be perceived as an advantage, but it comes with the drawback of intransparency on the optimization process and absence of challenging the results critically. Accepting the results of the optimization as 'given' and not questioning the meaningfulness of the user specifications can lead to results which are far away from an ideal, or even acceptable solution. Even for more experienced users, choosing a well suited setup for the specification of the optimization is not arbitrary.

Therefore, this work presents the R-package *VisBayesOpt*, which enables a broad visualization of different aspects of SMBO, obtained from model-based optimizations conducted with the package *mlrMBO*[6]. *VisBayesOpt* provides a visualization of the entire run, as well as a diagnostic tool for analyzing a single iteration of the SMBO. The work is outlined as follows. The next section describes the theoretical aspects of SMBO including a formal statement of the SMBO problem and elaborates on the important parts of such, the surrogate model and the acquisition function. Subsequently, existing visualization tools of packages from R and Python are reviewed and shortcomings of these pointed out. Afterwards the package *VisBayesOpt* is introduced. This covers an overview of optimization problems manageable by the package and an introduction of the Shiny application, which provides a user-friendly interface for the analysis of a *mlrMBO*-run.[1] Besides that, an exemplary diagnostic analysis is provided, which covers common problems in SMBO and how to approach and overcome these problems by the usage of the visualizations from *VisBayesOpt*.

[1] Formally the model-based-optimization-run is an object of class *MBOSingleObjResult* that is returned by the function *mbo()* from the package *mlrMBO*.

2 The Framework of Sequential Model-Based Optimization

This section introduces the Framework of SMBO. First, the optimization problem is formally stated and the algorithm, which guides the SMBO process, is presented. Subsequently, the central elements of SMBO, the surrogate model and the acquisition function is defined.

2.1 Problem Statement

In the context of optimization $f : \mathcal{X} \rightarrow \mathcal{Y}$ describes a black-box function which maps a p-dimensional search space $\mathcal{X}$ to an associated outcome space $\mathcal{Y}$. The search space vector is denoted as $\mathbf{x} = (x_1, \ldots, x_p)^\top \in \mathcal{X}$ while the observed output is $y \in \mathcal{Y}$. Search spaces can be either numeric $\mathcal{X} = \mathbb{R}^p$, bounded $\mathcal{X} \subset \mathbb{R}^p$ or categorical.[16][p. 8]

Specifying empirical observations, $(\mathbf{x}^{(i)}, y^{(i)})$ describes the i-th observation, thus a set of n observations is denoted as $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\} \in (\mathcal{X} \times \mathcal{Y})^n$. All observations are assumed to be realizations of independent and identically distributed (*iid*) random variables $(\mathbf{X}, \mathbf{Y})$ that follow a joint probability distribution $\mathbb{P}_{X,Y}$.

Given the fundamental definitions we subsequently state the optimization problem. Each SMBO considers a general optimization problem of the form

$$\mathbf{x}^\star = \min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) \tag{1}$$

where $\mathbf{x}^\star$ corresponds to the set of search space values which minimize the objective function f. Note that an equivalent maximization problem is obtained by changing the prefix of $f(x)$. To determine the optimum, SMBO carries out the steps outlined in algorithm 1.[17][p. 150], [6][p. 4-5].

Algorithm 1: BO Search Procedure

```
init Generate the initial design $\mathcal{D}_0 = (\mathbf{x}_0^{(i)}, y_0^{(i)})$ by sampling points
  $\mathbf{x}_0^{(i)}, i = 1, \ldots, n$ and evaluate $f$ at these points, which yields $y_0^{(i)}$. Use $\mathcal{D}_0$ to fit
  the initial surrogate $\hat{f}_0$. Set iteration $k = 0$.
while Termination criterion not met do
    Step 1: Propose $m$ new points $\mathbf{x}_{k+1}^{(i+j)}, j = 1, \ldots, m$ based on acquisition
      function in iteration $k + 1$ (For single proposal $m = 1$).
    Step 2: Evaluate the objective function at $\mathbf{x}_{k+1}^{(i+j)}$ yielding $y_{k+1}^{(i+j)}$ and augment
      the design with the new tuple $\mathcal{D}_{k+1} = \{\mathcal{D}_k, (\mathbf{x}_{k+1}^{(i+j)}, y_{k+1}^{(i+j)})\}$
    Step 3: Update the surrogate model $\hat{f}_{k+1}$ based on the new design $\mathcal{D}_{k+1}$
end
return Best solution $\mathbf{x}_{opt}$
```

In generating the initial design the user is faced with a trade-off between choosing too few points, not covering $\mathcal{X}$ well, and choosing to many points, using significant time to initially evaluate f.[6][p. 5] The proposal of new points is guided by maximizing the acquisition function (also called infill criterion) which balances exploration and exploitation. One prerequisite of SMBO is that the acquisition function is cheap to evaluate (compared to the objective function f), which is mostly the case when it has a closed form notation.[17][p. 150] Once the objective function f is evaluated with the newly proposed points $\mathbf{x}_{k+1}^{(i+j)}$ the design is augmented with the resulting tuple $(\mathbf{x}_{k+1}^{(i+j)}, y_{k+1}^{(i+j)})$. Following this, the surrogate $\hat{f}_{k+1}$ is updated based on the new design $\mathcal{D}_{k+1}$. The decision of choosing an appropriate surrogate is based on the structure of the input space $\mathcal{X}$. For $\mathcal{X} \subset \mathbb{R}^p$ Kriging, which is built on a Gaussian Process (GP), is an often applied method.[6][p. 5] The GP is easy to handle since it is entirely specified by its mean and covariance function for a given input $\mathbf{x}$.[2][p. 10-11] We will show this property later when formally introducing the GP. For mixed search spaces of numeric and categorical parameters random forests (RFs) are a suitable alternative for the surrogate model, which we will also discuss later on.[6][p. 5]

In the following the main elements of SMBO, the surrogate model and the infill criterion, are introduced. First, we present the general concepts and subsequently provide frequently applied models for both concepts in practical applications.

2.2 Surrogate Model

The surrogate model serves as a proxy for the real, unknown objective function and incorporates the entire information (of evaluations) present up to the current iteration $k+1$, i.e. $\mathcal{D}_{k+1}$. Most applications consider probabilistic surrogates due to their theoretical properties of offering an uncertainty estimation, which helps to balance the trade-off between exploration and exploitation.[2][p. 10] The later introduced package *VisBayesOpt* is geared to handling optimization problems from *mlrMBO*, thus the default surrogate models used in *mlrMBO* are introduced. For numeric only (including integer) parameter spaces *mlrMBO* (by default) uses a Kriging model (i.e. GP regression) while for mixed numeric-categorical parameter spaces a RF model is used.[2]

First, the GP regression is introduced from the function-space view, where we can think of a GP as defining a distribution over functions and inference is directly conducted in the space of functions.[16][p. 7] A GP is a collection of random variables which exhibit a joint gaussian distribution. It is entirely specified by its mean function $m(\mathbf{x})$ and its covariance function $k(\mathbf{x}, \mathbf{x}')$ and is formally defined as:[16][p. 13]

$$f(\mathbf{x}) \sim \mathcal{GP}\big(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\big) \tag{2}$$

where:

$$\begin{aligned} m(\mathbf{x}) \quad &= \mathbb{E}[f(\mathbf{x})] = \mathbb{E}[f(\mathbf{x}^{(i)})]_{i=1,\dots,n} \\ k(\mathbf{x}, \mathbf{x}') \quad &= \mathbb{E}\big[\big(f(\mathbf{x}) - m(\mathbf{x})\big)\big(f(\mathbf{x}') - m(\mathbf{x}')\big)\big] \\ &= \mathbb{E}\big[\big(f(\mathbf{x}^{(i)}) - m(\mathbf{x}^{(i)})\big)\big(f(\mathbf{x}^{(j)}) - m(\mathbf{x}^{(j)})\big)\big]_{i,j=1,\dots,n} \end{aligned}$$

The GP comes with the marginalization property which ensures that the examination of a larger set of variables does not change the distribution of the smaller one, i.e. if a $\mathcal{GP}$ specifies $(y_1, y_2) \sim \mathcal{N}(\mu, \Sigma)$ then it also specifies $y_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$.[3][16][p. 13] Having outlined the general setup of a GP we now move on to sampling from prior and posterior of a GP. Therefore we assume the mean function to be zero, i.e. $m(\mathbf{x}) = 0$.

First, we have a more detailed look at the covariance function since it specifies the distribution over the sampled functions. For most practical applications we need to bear in mind that we only observe noisy values $y(\mathbf{x})$ of the function $f(\mathbf{x})$, so we

[2] For default properties see documentation of *makeMBOLearner()* in package *mlrMBO*.
[3] Where Σ_{11} is a sub-matrix of Σ and $\mu = (\mu_1, \mu_2)^\top$.

can state the relationship as $y = f(\mathbf{x}) + \epsilon; \epsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$.[16][p. 16] Assume we have a set of observations $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)}) | i = 1, \ldots, n\}$ and a set of unobserved values $\mathcal{D}_\star = \{(\mathbf{x}_\star^{(i)}, y_\star^{(i)}) | i = 1, \ldots, n_\star\}$, we can write the observed search space values as $\mathbf{X} = (x^{(1)}, \ldots, x^{(n)})^\top$ and the unobserved test points by $\mathbf{X}_\star = (x_\star^{(1)}, \ldots, x_\star^{(n_\star)})^\top$.

Having introduced the notation of noisy observations, the general form of the covariance function over all function values $\mathbf{y} = (y^{(1)}, \ldots, y^{(n)})^\top$ is given by:[16][p. 16]

$$Cov\big(y^{(i)}, y^{(j)}\big) = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) + \sigma^2 \delta_{ij} \tag{3}$$

where:

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}$$

$\mathbf{I}_n$ n-dimensional identity matrix

Due to the assumption of noisy observations it is necessary to incorporate the variance term σ^2, of the random error ϵ, iff $\mathbf{x}^{(i)} = \mathbf{x}^{(j)}$. Using the relationship from equation (3), we can state the prior distribution of $\mathbf{y}$. Therefore we assume that f follows a Gaussian process with a mean vector of zeros and the pre-specified covariance matrix $\mathbf{K}$, which results from the chosen kernel function. The prior distribution of $\mathbf{y}$ is given by:

$$\mathbf{y} \sim \mathcal{N}\big(\mathbf{0}, \mathbf{K} + \sigma^2 \mathbf{I}_n\big) \tag{4}$$

where:

$$\mathbf{K} = \Big(k\big(\mathbf{x}^{(\mathbf{i})}, \mathbf{x}^{(j)}\big)\Big)_{i,j=1,\ldots,n}$$

Plotting the sampled values as a function of the inputs is known as *sampling from prior*. [2][p. 38] This enables us to state the joint distribution of the observed target values $\mathbf{y}$ and the unknown function values $\mathbf{f}_\star$ under the prior at the test points as:[16][p. 16]

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_\star \end{bmatrix} = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma^2 \mathbf{I}_n & k(\mathbf{X}, \mathbf{X}_\star) \\ k(\mathbf{X}_\star, \mathbf{X}) & k(\mathbf{X}_\star, \mathbf{X}_\star) \end{bmatrix}\right) \tag{5}$$

We can now state the conditional distribution for the test outputs $\mathbf{f}_\star$ by conditioning on the test inputs $\mathbf{X}_\star$. This yields the posterior equations for the GP regression with the posterior mean $\mathbf{m}_\star$ and the posterior covariance matrix $\mathbf{K}_\star$:[16][p. 16-17]

$$\mathbf{f}_\star | \mathbf{X}, \mathbf{y}, \mathbf{X}_\star \sim \mathcal{N}(\mathbf{m}_\star, \mathbf{K}_\star) \tag{6}$$

where:

$$\begin{aligned} \mathbf{m}_\star = &\quad \mathbb{E}[\mathbf{f}_\star | \mathbf{X}, \mathbf{y}, \mathbf{X}_\star] = k(\mathbf{X}_\star, \mathbf{X})[k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}_n]^{-1}\mathbf{y} \\ \mathbf{K}_\star = &\quad Cov(\mathbf{f}_\star) = k(\mathbf{X}_\star, \mathbf{X}_\star) - k(\mathbf{X}_\star, \mathbf{X})[k(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}_n]^{-1} k(\mathbf{X}, \mathbf{X}_\star) \end{aligned}$$

This formula also holds for the noise-free case, where $\sigma^2 = 0$. Having outlined the general setup for calculating priors and posteriors of the GP we get back to the kernel function which determines the distribution of the GP.

In general the kernel should state our assumptions about the link between the input space $\mathcal{X}$ and the output space $\mathcal{Y}$. Generally spoken, a kernel function transforms points in a way such that, if input space values are close to each other, the corresponding output values are close to each other too.[2][p.40-41] To see how this relationship is established in the functional form of a kernel, two common kernels, the squared-exponential kernel and the Matérn kernel are introduced. In general, any function k which transforms two arguments $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ into a scalar and satisfies the following two characteristics is a kernel function:

(i) symmetry: $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x}) \Leftrightarrow k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = k(\mathbf{x}^{(j)}, \mathbf{x}^{(i)})\ \forall i, j = 1, \ldots, n$

(ii) Positive semidefinite (PSD) Gram matrix $\mathbf{K} = \Bigg(k\big(\mathbf{x}^{(\mathbf{i})}, \mathbf{x}^{(j)}\big) \Bigg)_{i,j=1,\ldots,n}$

Bearing these properties in mind the *Squared Exponential (SE) kernel* function is defined as:

$$k_{SE}(\mathbf{x}, \mathbf{x}') = exp\Bigg(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2l^2} \Bigg) \tag{7}$$

where:

$\|\| \cdot \|\|$	Euclidean norm
l	Characteristic length-scale

The characteristic length-scale rescales any point $\mathbf{x}$ by $1/l$. A short length-scale makes function values only strong correlated when their input values are close to each other, while a large length-scale implies long ranged correlations.[4] One important property of

[4] We can determine the *closeness* among the search space values by calculating a distance measure (e.g. euclidean distance for numeric search spaces or Gower distance for mixed search spaces).

the SE kernel is its smoothness, as it is indefinitely differentiable.[5][2][p. 41] The second kernel function introduced is the *Matérn kernel* which uses the modified Bessel function K_ν[6] and is formally defined by:

$$k_{Mat}(\mathbf{x}, \mathbf{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(- \frac{||\mathbf{x} - \mathbf{x}'||\sqrt{2\nu}}{l} \right)^{\nu} K_\nu \left(- \frac{||\mathbf{x} - \mathbf{x}'||\sqrt{2\nu}}{l} \right) \tag{8}$$

where:

$\Gamma(\cdot)$	Gamma function

For $\nu \to \infty$ the Matérn kernel approaches the SE kernel. Now the GP regression is formally introduced. By now we can built a surrogate model for numeric search spaces, with a specified kernel function. Besides that, the sampling from prior was introduced which enables us to draw samples based on the chosen kernel. In addition the posterior equations were setup that allow us to incorporate our knowledge in the surrogate, once new observations are made. Now we move on to the RF regression to built surrogate models for mixed numeric-categorical parameter spaces.[2][p. 52]

The RF regression is an ensemble learning method that uses the principle of bagging, to sample from the dataset, and randomly select search space components based on which every tree of the forest is trained. The dataset is given by our design $\mathcal{D}$ of n observations. The RF regression comes with the ability to calculate the mean estimate and the standard deviation estimate based on the observations. Technically this can be done by calculating the mean and variance of the results generated from the individual trees. Using the estimates of the mean and the standard deviation, we can construct the surrogate model for y. Assume S to be the number of decision trees in the forest, $T_i(\mathbf{X})$ the output provided by the i-th decision tree. We can compute the variance estimate, as the empirical variance over all trees, by:[2][p. 52]

$$\mathbb{V}\left(\frac{1}{S} \sum_{i=1}^{S} T_i(\mathbf{x}) \right) = Cov\left(\frac{1}{S} \sum_{i=1}^{S} T_i(\mathbf{x}), \frac{1}{S} \sum_{j=1}^{S} T_j(\mathbf{x}) \right) = \rho\sigma^2 + \sigma^2 \frac{1-\rho}{S} \tag{9}$$

where:

[5] This property comes from the differentiability of the exponential function $exp(\cdot)$.
[6] For details on the modified Bessel function see [1][p. 374-377].

σ^2	maximum variance of all $T_i(\mathbf{x})$
$\rho\sigma^2 = Cov(T_i(\mathbf{x}), T_j(\mathbf{x}))$	maximum covariance of all combinations of $T_i(\mathbf{x}), T_j(\mathbf{x})$

Equation (9) shows that the variance of the RF regression is proportional to σ^2 and ρ and is increasing with the size S of the RF. The mean estimate can also be calculated by its empirical equivalent. Besides these properties, the computational efficiency of RF regression, compared to GP regression, should be highlighted. For optimization problems with a large number of search space components RF does not need to invert a the kernel matrix, like in GP regression, and parallelization of the RF regression decreases the time spent on the computation of the model.[2][p. 51-53]

As of now we introduced the surrogate models for numeric and mixed parameter spaces in SMBO. With the surrogate we can incorporate all information known at a certain iteration into the model. This enables us to derive an estimate for the posterior mean function $\hat{\mu}(\mathbf{x})$ and an uncertainty estimate based on the estimated variance $\hat{s}(\mathbf{x})$ by the surrogate. Now we turn towards the question where the unknown function f should be evaluated next. This is done by the acquisition function. We will focus on acquisition functions for single-point-proposals ($m = 1$)[7] and for simplicity assume noise-free objective functions.

2.3 Acquisition Function

The proposal of the new point to evaluate the objective function next is based on the trade-off between exploration and exploitation which is handled by the acquisition function (infill criterion). *mlrMBO* offers a variety of acquisition functions.[8] For the subsequent section we introduce the common infill criteria Probability of Improvement (PI), Expected Improvement (EI) and the Lower Confidence Bound (LCB).

To start with, the PI is defined by:

$$PI(x) = \mathbb{P}(f(\mathbf{x}) \leq f_{min}(\mathbf{x}) + \lambda) = \Phi\left(\frac{f_{min}(\mathbf{x}) - \hat{\mu}(\mathbf{x}) - \lambda}{\hat{s}(\mathbf{x})}\right) \tag{10}$$

where:

[7] Acquisition functions for multi-point-proposals can be found in [6][p. 10].
[8] To inspect the available acquisition functions see the help page of the function *MBOInfillCrit*.

$f_{min}(\mathbf{x})$	Best value of objective function observed so far
$\hat{\mu}(\mathbf{x}), \hat{s}(\mathbf{x})$	Estimate for mean and standard deviation of surrogate model
$\lambda \geq 0$	Parameter to control exploration/exploitation. $\lambda = 0$ is pure exploitation
$\Phi(\cdot)$	Cumulative distribution function of normal distribution

For the parameter λ we notice that low (high) values of λ favor exploitation (exploration). To derive the next point $\mathbf{x}^{(n+1)}$[9] to sample, the PI is maximized, i.e.

$$\max_{\mathbf{x} \in \mathcal{X}} PI(\mathbf{x}) \tag{11}$$

One of the main disadvantages of the PI is the fact that the magnitude of the improvement is not considered, once a value is assigned to a new point.[2][p. 57-58] This weakness can be overcome by the next infill criterion, the EI.

The EI measures the expectation of improvement w.r.t. the predictive distribution of the surrogate model and is formally defined as:

$$EI(\mathbf{x}) = \begin{cases} (f_{min}(\mathbf{x}) - \hat{\mu}(\mathbf{x}) - \lambda)\Phi(Z) + \hat{s}(\mathbf{x})\phi(Z) & \text{,if } \hat{s}(\mathbf{x}) > 0 \\ 0 & \text{,if } \hat{s}(\mathbf{x}) = 0 \end{cases} \tag{12}$$

with:

$Z =$	$\begin{cases} \frac{f_{min}(\mathbf{x}) - \hat{\mu}(\mathbf{x}) - \lambda}{\hat{s}(\mathbf{x})} & \text{,if } \hat{s}(\mathbf{x}) > 0 \\ 0 & \text{,if } \hat{s}(\mathbf{x}) = 0 \end{cases}$
$\phi(\cdot)$	Probability distribution function of normal distribution

The first term in equation (12) increases if the predictive mean of the surrogate decreases, while the second term increases if the uncertainty in the surrogate model increases. This shows how EI automatically balances exploration and exploitation while the degree of both can be controlled by the choice of λ. Visually spoken, the acquisition function tends to flatten with higher values for λ, in an extreme case the function can get close to random search.[10] The proposal of a new point $\mathbf{x}^{(n+1)}$ is derived by maximizing the EI [2][p. 58-60], i.e.

$$\max_{\mathbf{x} \in \mathcal{X}} EI(\mathbf{x}) \tag{13}$$

[9] Note that we only consider single-point-proposal, i.e. $j = 1$.

[10] See equation (12), in which the first term vanishes for high values of λ.

To effectively manage the trade-off between exploration and exploitation we introduce the LCB.

The LCB is geared at managing the proposal of points in a minimization problem[11] and is given by:

$$LCB(\mathbf{x}) = \hat{\mu}(\mathbf{x}) - \lambda \hat{s}(\mathbf{x}) \tag{14}$$

The proposal of a new point $\mathbf{x}^{(n+1)}$ is attained by minimizing the LCB[12][2][p. 60-62], i.e.

$$\min_{\mathbf{x} \in \mathcal{X}} LCB(\mathbf{x}) \tag{15}$$

Having outlined different infill criteria, we will turn towards the concrete proposal of new points. This task is accomplished by the infill optimizer, that searches for the point $\mathbf{x}$, which yields the best infill value. Compared to the evaluation of the objective function, the infill function is cheap to evaluate. *mlrMBO* uses the method of so called *focus search*, which handles a broad variety of search spaces, among others numeric, categorical and mixed search spaces. For the detailed *focus search* algorithm implemented in *mlrMBO* we refer to the published paper in accordance with the package *mlrMBO*.[6][p. 6-7]

[11] For a maximization problem the equivalent upper confidence bound needs to be considered.
[12] In the case of upper confidence bound maximization is required.

3 Visual Diagnostics for Bayesian Optimization

In this section the *VisBayesOpt* package is introduced. First, we review existing visualization tools in R and Python and motivate *VisBayesOpt* by shortcomings and improvements in the packages under review. Subsequently, we provide an overview of *VisBayesOpt* and introduce the Shiny application that comes with the package. The Shiny application provides a user friendly interface to the visualizations from the package. Finally, an exemplary analysis is conducted, which demonstrates how to use the package and to which extend the user can add value to its understanding of SMBO in general and the refinement of the optimization run.

Before introducing the package, the target user group and value add of *VisBayesOpt* is stated. As outlined in the Introduction there are various scientists making use of SMBO in their work. The target group of *VisBayeOpt* should be any scientist from one of the various disciplines outlined in the introduction, an in-depth knowledge on the statistical foundations and mathematical implementations of SMBO is not required. For this user group the Shiny application adds a substantial value since it facilitates the analysis of an SMBO run with a user-friendly interface. Besides that more experienced users can use the package to get an overall and handy insight into a SMBO run. For sure, advanced users can produce each single visualization from *VisBayesOpt* on their own, but due to reasons of time and complexity (especially with handling a high number of SMBO runs) such an in depth review is mostly not carried out.
The value add for both user groups is manifold. First, *VisBayesOpt* provides an enhancement on the understanding of the optimization process, removing the intransparency (complexity) of the optimization process. This enhancement is especially important for the less statistical oriented user group. In this case the tool helps to facilitate the 'what' the optimizer does thus preventing to take the results from the SMBO run as given without questioning the process of the optimization. The second value add comes by detecting mis-specifications based on the provided visualizations. Among others, such mis-specifications consist of to wide/narrow input spaces, too tight stopping rules (while results still improve) or an inappropriate surrogate model. In the exemplary conducted analysis, later in this section, we show how to detect these mis-specifications by the help of *VisBayesOpt*.

3.1 Review of Existing Visualization Tools

First, we review existing tools for visualizations of SMBO. This review includes the R packages *tune* and *mlrMBO* and the Python packages *hyperopt* and *scikit-optimize*. Besides that we looked at the Python packages *smac* and the newly published package *BoTorch*, but these do not, to the knowledge of the author, include any built-in visualization functions. All visualizations presented in the following review can be reproduced by the provided R/Pyhton code in the github repo.[13] The general procedure of the review is the same for all packages and proceeds as follows: First, we take a look at the possible visualizations that can be produced with the plot functions of the package. Afterwards we discuss improvements of the plots. At this stage we will also mention the respective plot classes of *VisBayesOpt*, which overcome these shortcomings. We will not explicitly state additional visualizations, that might be helpful for a broader understanding of the SMBO each time, since most packages come with a limited number of possible plots.

3.1.1 R packages tune and mlrMBO

For R we have a look at the two packages *tune* [13], an extension to the *caret* package for bayesian optimization, and *mlrMBO* [5], a package focused on model based optimization based on the *mlr* package.

In *tune* we can visualize results from a SMBO in three types of plots.[14] The function *autoplot.tune_results()* is the central plot function of the package and enables all plots. The plot type is specified with the *type* argument, i.e. $type \in \{performance, marginals, parameters\}$. Figure 1 shows the three possible visualization (combined as facets) of the tuning result. The left facet of the first plot shows the accuracy over the iterations, calculated as out-of-sample estimates. Besides that the Area Under the Curve (AUC) of the model is plotted which measures the overall predictive capability of the model. The second visualization (*type = "marginals"*) plots the same performance measures over the search space components. In the third visualization (*type = "parameters"*) the value of each evaluated search space component is plotted over the

[13] Please see repo for final submission of this bachelors' thesis.

[14] The function *tune_bayes()* returns an object of class *tibble* with all information from the bayesian optimization run.

single iterations. The middle plot of figure 1, for the accuracy over the parameter values (*type = "marginals"*), could be extended by including the parameter values sampled from the specified input space. This could provide us with an intention, how the optimizer

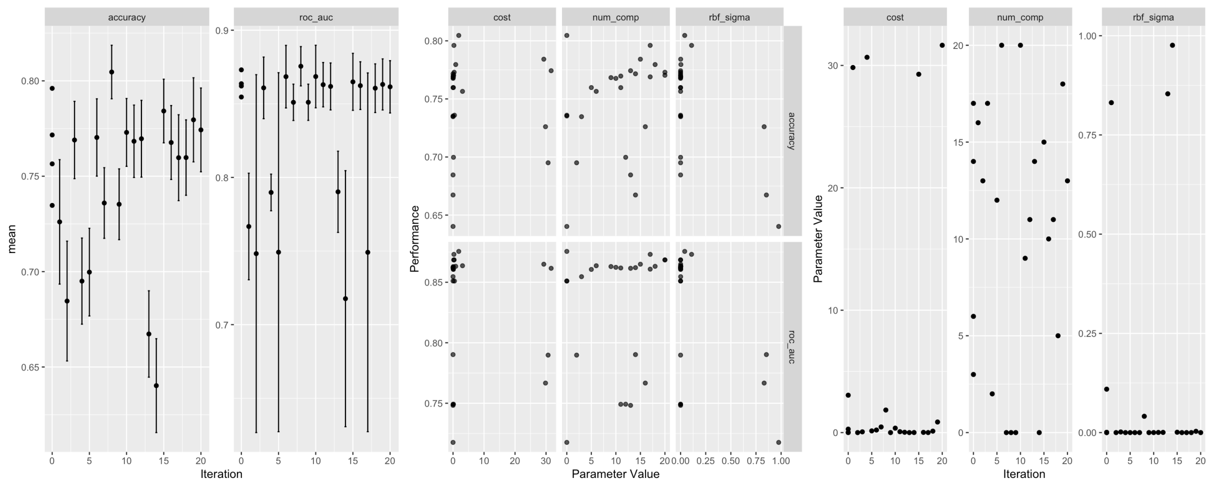


Figure 1: Visualization R package *tune*. Shows all possible types *performance, marginals, parameters* from left to right. *Performance* plots the model performance (i.e. accuracy and AUC) over each iteration. *Marginals* plots the model performance against each value of the search space components. *Parameters* plots the value of each search space component over the iterations. The visualization is based on a svm model for a classification task on the *cells* dataset from the *modeldata* library.[15]

proceeds, compared to a random search strategy. Besides that, it would give us an insight into the sampling of the initial design, since we could notice transformations of the search space components, if specified by the user. We will see, that *MboPlotInputSpace* and *MboPlotDependencies* from *VisBayesOpt* address these shortcomings. The third plot of Figure 1 (*type="parameters"*) lacks the information on the target. If the target value would be included (as a third dimension, e.g. as color) one could distinct between exploration (target has probably a worse value) and exploitation (target has probably a better value) and thus interpret the values of the search space components in light of the outcome y. This shortcoming is considered by the function *MboPlotSearchSpace* in *VisBayesOpt*.

Next we take a look at the package *mlrMBO*, which offers the three plot functions *plot.OptState(), plotExampleRun()* and *plotMBOResult()*. Figure 2 shows an exemplary

[15] Own illustration based on R.

output of *plotMBOResult()*. While the plot in the top visualizes the target variable over the search space, the lower plots show how the target and search space evolves over each iteration. Besides that the acquisition function is plotted over the iterations. In each single plot the points from the initial design are separated from the points during the iterations and the last proposed point is highlighted. A possible enhancement for the *plotMBOResult()* would be a combination of the search space components, the target and the iterations. This kind of visualization is provided by the class *MboPlotSearchSpace* of *VisBayesOpt*. Besides that the surrogate could be included in the upper plot which would provide the user with an intention on the "best guess" between the evaluated points. Besides these specific enhancements a few overall improvements on the visualization in *mlrMBO* are subsequently discussed.

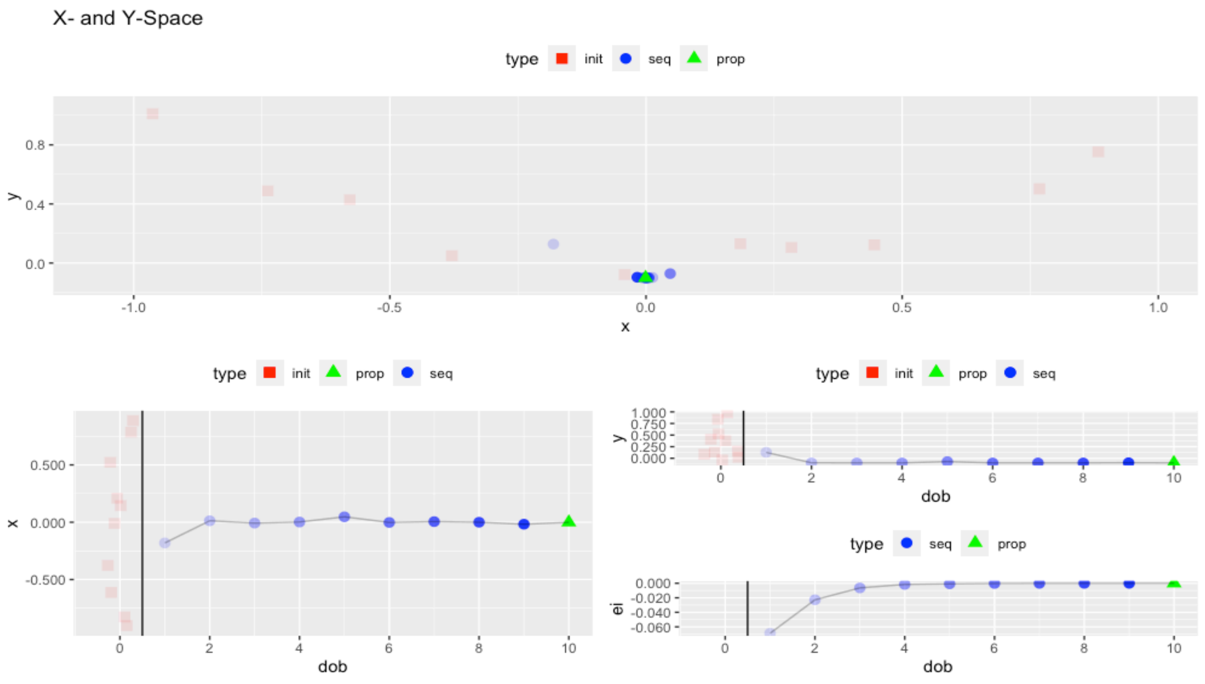


Figure 2: Visualization R package *mlrMBO*: plotMBOResult. In the plot on top the input and target variable is plotted. In the lower plot the search space component x, target y and infill criterion ei is plotted over the iterations.[16]

The package does not provide a function for visualizing the surrogate and acquisition function in a single iteration on an already performed run since *plotExampleRun()* requires an input from the specific function *exampleRun()* and does not handle objects from the central optimization function *mbo()* of the package. This limits the value add for practical

[16] Own illustration based on R. Example taken from mlrMBO examples.

users, as they would need to re-run the SMBO with the *exampleRun()* function, just for visualization purposes. Besides that all visualizations are limited to a 2 dimensional search space.[17] This limitation is especially severe for practical users, as most practical problems require higher dimensional search spaces. *VisBayesOpt* overcomes both problems, as it entirely operates on the *final.opt.state*[18] returned by the function *mbo()*. Besides

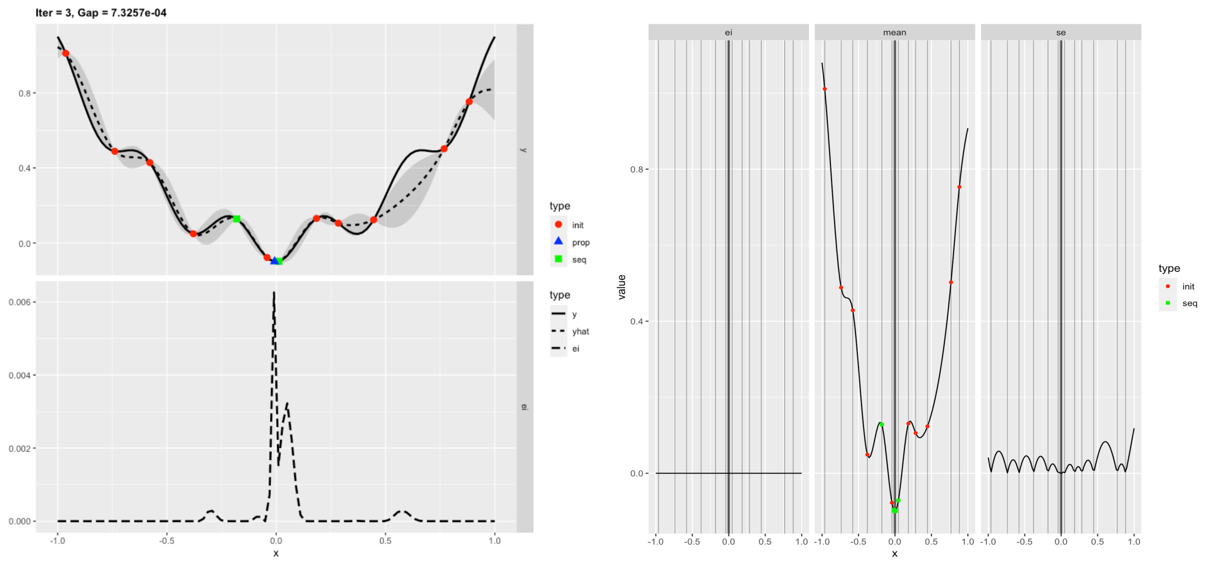


Figure 3: Visualization R package *mlrMBO*: plotExampleRun, plot.OptState. The left plot shows a specified iteration (iter=3) of the example run with the surrogate model in the top and the acquisition function in the bottom. The right plot visualizes the final optimization state with the acquisition function, the mean and the standard deviation (from left to right) over the search space.[19]

that *VisBayesOpt* also covers higher dimensional search spaces of numeric and discrete parameters for the visualization. In the next part of this chapter we focus on packages in Python.

3.1.2 Pyhton packages hyperopt and scikit-optimize

Subsequently, we take a look at the available visualizations from the Python packages hyperopt[20] and scikit-optimize[21]. All figures provided are set up in accordance with

[17] The visualizations for a 2 dimensional search space are not provided in the review.
[18] Object from class *OptState*.
[19] Own illustration based on R. Example taken from mlrMBO examples.
[20] See github package hyperopt.
[21] See github package scikit-optimize.

examples from the respective github page of the package and are modified where necessary.

To start with, the *hyperopt* package offers three main plot functions for the visualization of the SMBO result, *main_plot_history(), main_plot_vars()* and *main_plot_histogram()*. For the SMBO we specified an artificial 3 dimensional test function $z = \sqrt{x_1^2 + x_2^2}$ with parameters x_1, x_2 as inputs (search space components). Since *hyperopt* offers implemented functions for specifying the distribution of the input space we choose x_1 to be lognormal with $\mu = 0, \sigma = 0.5$ and x_2 uniform on the interval $[-6, 6]$. Figure 4 shows the available visualizations from *hyperopt*, based on the SMBO result. The first plot shows the true result y over each iteration.[22] To get a better understanding of the overall improvement of

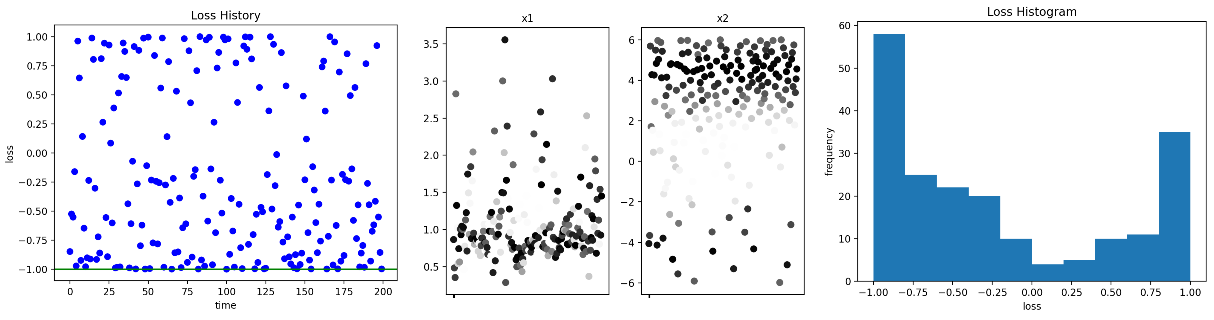


Figure 4: Visualization Python package *hyperopt*. From left to right: the first plot shows the evaluated output values $y^{(i)}$ (axis label 'loss') at each iteration i (axis label 'time'). The green line specifies the best evaluated output of the run (here -1.0). The second wrapped plot shows the values for each search space component (y-axis) over the iterations (x-axis). The color corresponds to the difference between the evaluated output value in the iteration and the best evaluated output, with white as largest and black as closest distance. The third plot shows a histogram of the results over all iterations for the respective losses.[23]

the optimization it might be helpful to show the minimum cumulative value of the output y over the iterations. This would enable us to see if the optimizer still improves in later iterations, or if it already converges in early iterations. The class *MboPlotProgress* from *VisBayesOpt* provides this type of plot.

The second plot shows the value of each search space component (y-axis) over the iterations (x-axis).[24] This plot might be improved by a pairwise comparison of the search

[22] The default axis label might be misleading in this plot since it shows y and not $\hat{y} - y$ which we would consider as *loss*.

[23] Own illustration based on Python. Example from hyperopt is modified where necessary but general setup taken from hyperopt examples.

[24] Label for y-axis and x-axis not provided by plot function; information based on source code of *hyperopt*.

space components in a matrix plot. This would enable the user to identify if search space components follow a certain pattern among each other. *VisBayesOpt* offers such a plot by the class *MboPlotDependencies*.

Next we take a look at the Python package *scikit-optimize*, which offers the greatest extend of modern visualizations for SMBO found within the scope of this review. It also implements visualizations for higher dimensional search spaces via Partial Dependence Plots (PDPs), known from Interpretable Machine Learning (IML).[25] In total *scikit-optimize* offers the five functions *plot_evaluations(), plot_objective(), plot_convergence(), plot_regret()* and *plot_gaussian_process()*.

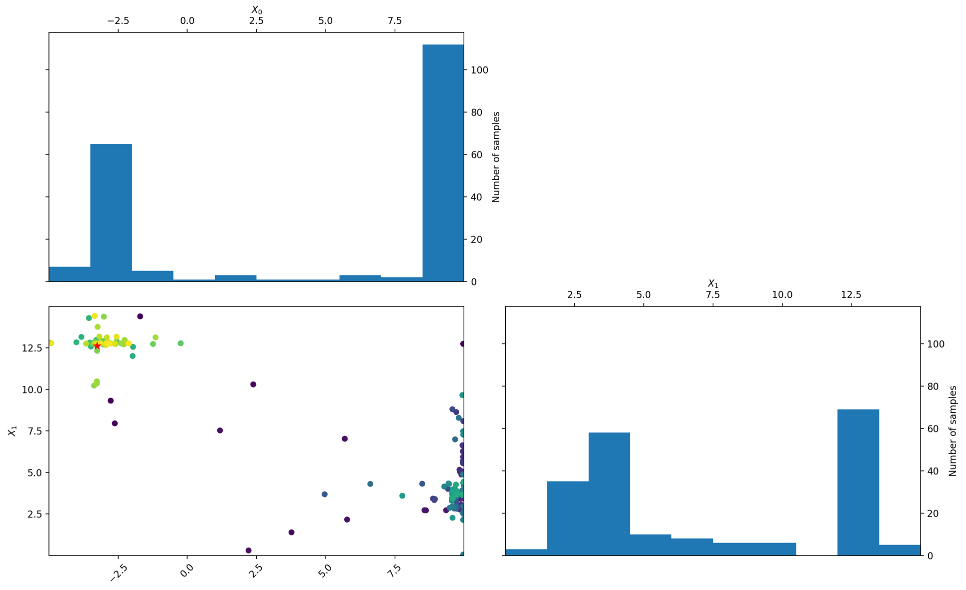


Figure 5: Visualization Python package *scikit-optimize*: plot_evaluations. Shows a plot matrix with histograms on the diagonal for the respective search space variables (x_0, x_1) and their combined scatter plot. The color of the scatters correspond to the iteration in which the points were sampled, from early iterations (dark) to late iterations (light). The combination of the search space components with the best target value of the run is highlighted with a red star.[26]

For the optimization we use the three dimensional artificial Branin test function and define the input space $x_0 \in [-5, 10], x_1 \in [0, 15]$ and limit the number of evaluations to 200. Figure 5 shows the output of *plot_evaluations()* for the SMBO run. The visualization points out the dependence between the search space variables and is thus well suited to

[25] For more details on PDPs see chapter 5.1 of [15].

[26] Own illustration based on Python. Example from scikit-optimize is modified where necessary but general setup taken from scikit-optimize examples.

serve its purpose. One improvement may be, to include a possibility to color the points with the value of the output variable y. This would enable us, getting an insight into the dependence between the search space components and the output, too. In *VisBayesOpt* we provide the class *MboPlotDependence*, which enables to color the points either by the iteration or by the value of y.

Figure 6 shows the visualization of the objective function as a PDP of the single search space components and the effect on the output in the contour plot. This plot enables

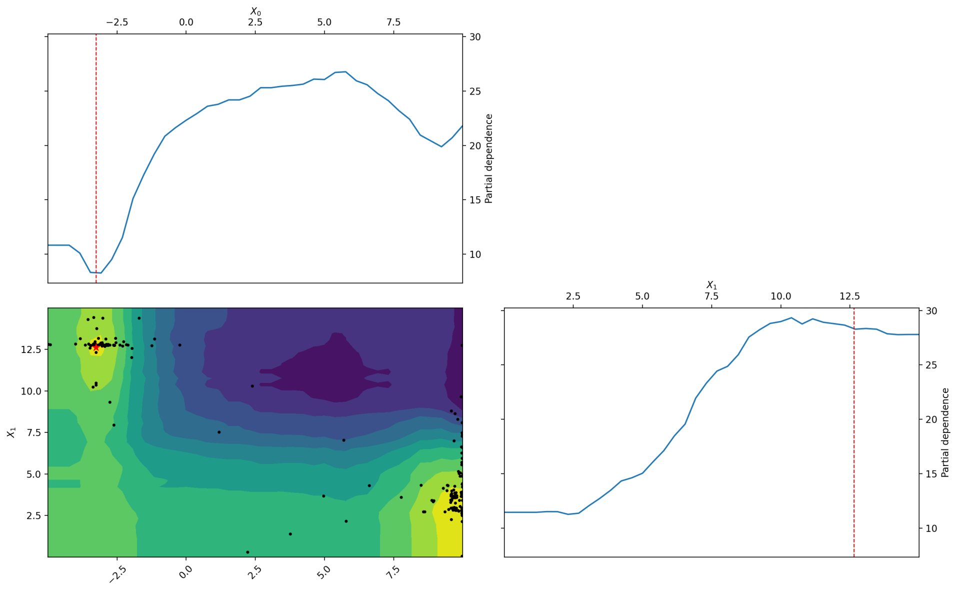


Figure 6: Visualization Python package *scikit-optimize*: plot_objective. Shows a matrix plot with effects of the single search space component (x_0, x_1) on the objective function on the diagonal. The plot below the diagonal shows the effect on the objective function when varying the two search space components (x_0, x_1). The scatters show the points evaluated during the iterations. The red star highlights the minimum of the objective function y evaluated during the SMBO run.[27]

to extract all relevant information on the dependence between search space components and output and does not lack any missing information in that respect. Figure 7 shows the *plot_convergenge()* and the *plot_regret()*. In the convergence plot we can see the improvement during the optimization, thus giving an intention of where the optimizer still improves and when there is no/only minor improvement (i.e. convergence). The cumulative regret plot shows the cumulative difference between all evaluated search space components and the true optimum value of the output. This leads to a monotone

[27] Own illustration based on Python. Example from scikit-optimize is modified where necessary but general setup taken from scikit-optimize examples.

increasing function, which increases more if the prediction $f(\mathbf{x}^{(i)})$ in iteration i is still far away from the optimum (e.g. in first iterations of plot) or if the optimizer still explores a lot (which is also possible in later iterations). Both visualizations help the user to understand if the optimizer converges or still improves and thus provide a nice overview on the optimization. *VisBayesOpt* does not provide a plot of the cumulative regret as the true value of the target is rarely (if at all) known in practical setting of SMBO. All in

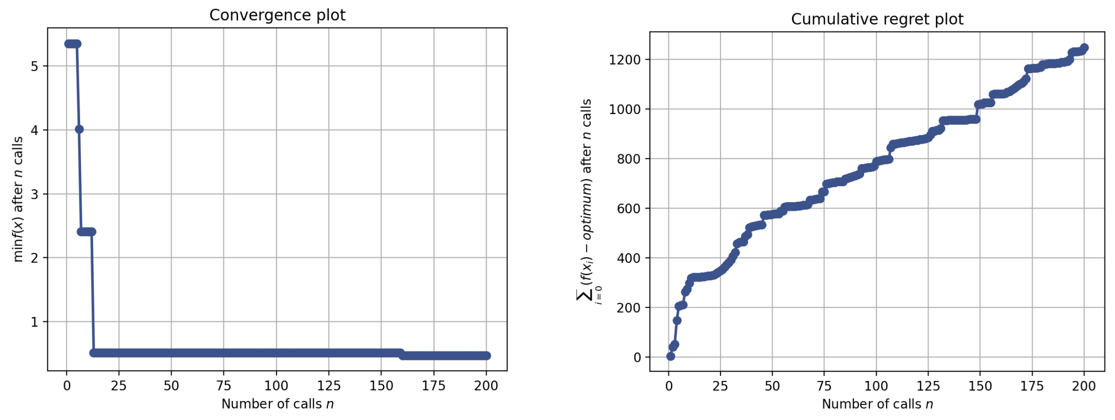


Figure 7: Visualization Python package *scikit-optimize*: plot_convergenge, plot_regret. From left to right: The first plot shows the minimum value of the objective function over the single iterations. The second plot shows the cumulative difference between the objective function $f(\mathbf{x}^{(i)})$ of iteration i and the true minimum value of the objective function (optimum).[28]

all we found the implemented visualizations in *scikit-optimize* already profound in their informative value.

Until now we have seen an overview of the existing environment of visualizations in the context of SMBO in the programming languages R and Python. We found the most packages offer only a limited number of visualizations or are subject to constraints of low dimensionality. To overcome these issues we now introduce *VisBayesOpt* which covers most of the seen visualizations in the review section and enhances the understanding of SMBO by providing additional insights. It also incorporates the suggested improvements of the review section.

[28] Own illustration based on Python. Example from scikit-optimize is modified where necessary but general setup taken from scikit-optimize examples.

3.2 The VisBayesOpt Package

In this section we introduce *VisBayesOpt*[29], the package implemented as part of this thesis. First, we give a short introduction to the package which is followed by an explanation of the Shiny application. Subsequently, we demonstrate the usage of *VisBayesOpt* by analyzing an example from a practical ML-context by methods of *VisBayesOpt*.

3.2.1 Introduction to VisBayesOpt

VisBayesOpt aims to visualize SMBO runs conducted with the package *mlrMBO*. It is implemented in a modern *R6* class design.[30] *R6* objects enable object-oriented programming, which is usually not the focus of R.[7][p. 3] In *VisBayesOpt* this object-oriented programming enables us to incorporate all public plot methods into our Shiny application, standardizing the way user interfaces (UIs) are generated. *VisBayesOpt* takes a *final.opt.state*[31] as input to initialize a new instance of the *R6* class object. *MboPlot* is the main class of which all other classes inherit.

Problem	Feasibility for *VisBayesOpt* under current implementation
Objective function	Real valued (no mixed-space)
Proposal of new points	Single proposal (no multi-point proposal)
Search space	Numeric and discrete search spaces

Table 1: Manageable problems for *VisBayesOpt*. Table shows the characteristics of optimization problems manageable for *VisBayesOpt* under the current implementation.[32]

To start with table 1 outlines the characteristics of supported optimization problems of *VisBayesOpt*. At a first glance some constraints (e.g. no mixed-space objective functions manageable) might limit the number of users but due to the modular setup of *VisBayesOpt* the constraints can be unset by expanding the plot classes. Besides handling the outlined problems from table 1 the package provides the general setup of the infrastructure and ideas of different possibilities for appropriate visualizations. Table 2 gives an overview of the classes implemented in *VisBayesOpt* with a short description on their *plot()* functions.

[29] See repository of *VisBayesOpt*.
[30] For details see *R6* package.
[31] Is of class *OptState* and incorporates all information from a *mbo()* run.
[32] Own illustration based on *VisBayesOpt*.

For a more detailed introduction to *VisBayesOpt* and how to use the package please see the readme file in the repository[33] of the package. It gives a full work-flow for handling the plot classes and how to adjust plot-specific parameters.

Plot class	Description of *$plot()* function
MboPlotDependencies	Matrix of pairwise scatterplots of the values of the search space components
MboPlotDistToNeighbor	Plots the Gower distance between the search space components
MboPlotEstimationUncertainty	Plots the degree of uncertainty in the estimation of new points
MboPlotFit	Plots the R-squared over the iterations and the predicted output $\hat{y}$ against the true output y
MboPlotInputSpace	Plots the histogram over the evaluated values of the single search space components
MboPlotOptPath	Plots the surrogate for a chosen iteration
MboPlotProgress	Plots the cumulative minimum value of the objective function $f(\mathbf{x})$ at the design points
MboPlotRuntime	Plots the split of the time spent during the optimization on the different tasks
MboPlotSearchSpace	Plots the values of each search space component evaluated by the optimizer over the iterations

Table 2: Plot classes of *VisBayesOpt*. Table shows the plot classes implemented in the package *VisBayesOpt* and gives a short description on the *plot()* function of each class.[34]

Besides the overview in table 2, appendix 1 provides a Unified Modeling Language (UML) diagram of the classes and how they relate to each other.

3.2.2 Shiny Application

VisBayesOpt comes with a Shiny application which can be run with the function *runAppLocal()*. The app is organized as follows: The *Setup* tab lets us upload any *final.opt.state* file from a *mlrMBO* run from a local directory. Once a run is uploaded a summary of the specifications of the run is displayed at the main panel. Another functionality in the tab is the export-plot button, which enables the user to export the

[33] See repository of *VisBayesOpt*.
[34] Own illustration based on *VisBayesOpt*.

last plot viewed in the application to a local directory. The tab *Visualize mlrMBO Run* contains all plots related to the overall run, while the tab *Diagnostic Tool for Single Iteration* is aimed to inspect a specified iteration in detail. Both tabs have a sidebar panel where we can select different parameters of the plots.[35] The sidebar enables the user to easily modify all parameters of the plots, providing a superior user experience, especially to unfrequent users of R. All plots come with a description section below, which explains the plot under review and guides the user on how to interpret the results and how typical patterns (of failure, improvements) look like. Having this guidance right at hand, the user can leverage the full value of *VisBayesOpt*.

The plots in the following section are generated with the functions implemented in the package, even though they are all part of the application too. We will now turn from the generalized context of SMBO towards a specific application in the context of ML.

3.2.3 Exemplary Diagnostic Analysis

In this section we use *VisBayesOpt* to analyze two specific *mlrMBO* runs. The analysis aims to highlight patterns which result from the different specifications of the models.

We build our analysis on the public available pid-task[36], which aims to classify Indian patients with several characteristics regarding their predisposition of having diabetes.[14][p. 29] For this classification task we use eXtreme Gradient Boosting (XGBoost), a tree boosting model which grows ensembles of trees by the technique of gradient boosting.[8][p.786-787] The aim of our SMBO is thus optimizing the various parameters (also called hyperparameters in the context of ML) of the XGBoost model. The following list states the hyperparameters and gives a short description on their task during the XGBoost training:[10][p 52-54]

- *nrounds*: maximum number of boosting iterations
- *eta* $\in [0, 1]$: tree parameter; contribution of each tree when added to the current approximation
- *max_depth*: tree parameter; maximum depth of a tree

[35] Note that the specifications in the sidebar panel do not affect all plots; please see the *Modifications* bullet of the description section in the application, to see which parameters belong to each plot. This layout comes due to the automatic generated UIs and might be enhanced to also display sections.
[36] For pid-task and various other example tasks see mlr example-tasks.

- *colsample_bytree*: tree parameter; subsample ratio of columns when constructing each tree
- *lambda*: boosting parameter; L2 regularization term on the weights
- *alpha*: boosting parameter; L1 regularization term on the weights
- *subsample* $\in [0, 1]$: ratio of the training instance, i.e. subsample=0.5 would mean that XGBoost uses only half of the data to grow trees.

We transform the parameter *lambda*[37] of the XGBoost model by taking the power function 2^{lambda}. This artificial transformation affects the random sampled initial design and we would like to see the effect of such transformation in our later analysis.

Characteristic	model1	model2
Infill Criterion	Confidence bound	Confidence bound
Infill Criterion Parameter	cb.lambda = 0.5	cb.lambda = 2.0
Optimization Direction	minimize	minimize
Surrogate-Model	Kriging	Kriging
Search Space	nrounds, eta, max_depth, gamma, colsample_bytree, lambda, alpha, subsample	nrounds, eta, max_depth, gamma, colsample_bytree, lambda, alpha, subsample
Number of Objectives	1	1
Multi-Point Proposal	1	1
Maximum number of Evaluations	200	200
Runtime [Minutes]	7.11	6.82
Minimum y	0.240	0.243

Table 3: MboSummary: model1, model2. The table shows the output of the function MboSummary$getMboSummary().[38]

Specifying the characteristics of the SMBO, we choose the GP (also known as *Kriging*) as surrogate and the LCB acquisition function. The 2 examples vary in the way such that lambda is chosen as $\lambda_1 = 0.5$ for model one and $\lambda_2 = 2$ for the second model. According to equation (14) the first model exploits, while the second explores, more frequent. We set the maximum number of evaluations to 200. We subsequently refer to the two models as *model1* and *model2*. The resulting *final.opt.state* objects can also be inspected in the repo of the package in the *test-data* thus the subsequent analysis can be followed by running

[37] Not to be confused with the λ of the acquisition function.
[38] Own visualization based on examples available in repo of *VisBayesOpt*.

the Shiny app and accessing the data from the repo.[39] The script for the general setup of the subsequent analysis can be accessed in the submission repository.[40]

Starting with the analysis, we first generate a summary of the models with the *MboSummary* class and use *MboShiny* to generate a table of these characteristics which is provided in table 3. The summary is more important for a practical user who has various *mlrMBO* runs on his local machine and thus gets a short wrap up of the specifications of the chosen model.

Going forward, we look at the plots of *MboPlotProgress* which shows the cumulative minimum value of the objective function$f(\mathbf{x})$ at the design points $\mathbf{x}$ after n iterations. Figure 8 depicts the plots for each of the two models under review. We can see that both

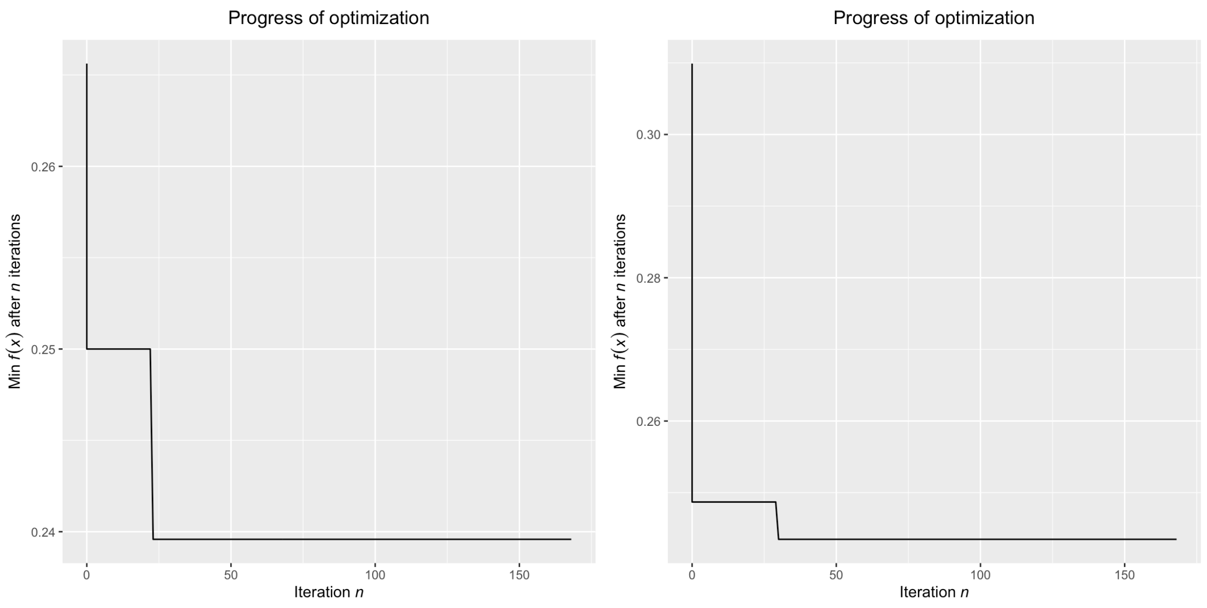


Figure 8: MboPlotProgress: model1, model2. The left (right) plot shows the cumulative minimum value of the objective function of model1 (model2) after n iterations.[41]

models converge within the first 30 iterations. For model1 (left plot) we can see that the optimizer finds a minimum cumulative value of $f(\mathbf{x})$ which is below that of model2. This behavior might results from the chosen tuning parameter λ which lets model1 exploit more than model2. Since the minimum cumulative value of both plots does not decrease beyond

[39] See test-data in github repository.
[40] Example script provided in submission-repository.
[41] Own illustration based on R. Example provided in submission-repository.

iteration 20 to 30 the optimizer might have converged. If so, it seems only of limited use, compared to additional computation cost, to increase the overall number of iterations of the optimization run. Since we now have a general intention on the convergence of the optimization we turn towards analyzing the input space in more detail.

For this purpose figure 9 shows the input space, generated by the plot function of class *MboPlotInputSpace*, for the selected search space components[42] *colsample_bytree, eta, lambda, nrounds*. The figure shows the sampled values during the optimization run (*entire optimization run*) as well as the sampling distribution of the initial design (*init design sampling distribution*). Comparing the entire optimization run to the initial

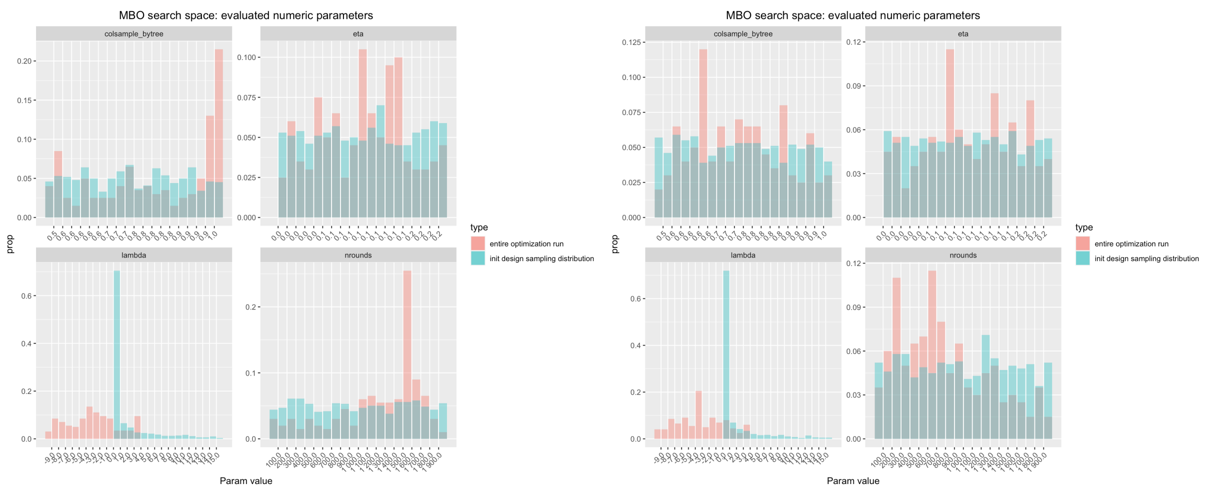


Figure 9: MboPlotInputSpace: model1, model2. The left (right) plot shows the histogram of the search space components *colsample_bytree, eta, lambda, nrounds* for model1 (model2). The *init design sampling distribution* shows artificially sampled values according to the input space, considering possible transformations of the input space. The *entire optimization run* shows the values actually evaluated by the optimizer.[43]

design sampling distribution for model1 we instantly note that the transformation of the parameter *lambda*, by 2^{lambda}, leads to large values around zero, while the optimizer is searching more frequently for negative values. The initial sampled design, which considers the transformation, might thus not be sufficiently covering the search space domain. By comparing both overlaid histograms, we can see how SMBO proceeds compared to a random search with the specified transformation. Looking at the parameter

[42] In the context of ML, search space components are also called features.
[43] Own illustration based on R. Example provided in submission-repository.

colsample_bytree reveals another pattern which often comes with the user specification of the search space. The optimizer frequently searches at the upper boundary of the search space, around one, which might be a sign that the value for *colsample_bytree*, leading to the minimum of the objective y, might be found above one. Thus in a second run we could adjust the upper boundary of *colsample_bytree* upwards. Comparing the plots of *colsample_bytree* between model1 and model2 we see the impact of the exploitation. While model1 exploits more, leading to frequent evaluations around one, model2 explores more, thus the limitations of the space seems not to affect the SMBO run with a higher degree of exploration. A similar pattern can be detected for the feature *nrounds*. We can see that values of below 500 are rarely evaluated by the optimizer, which might justify to limit the domain to a value of 500 at the lower boundary.

Next we take a look at the search space over the iterations, i.e. the proposal of new points during the SMBO run. Figure 10 shows the plot of *MboPlotSearchSpace*. In this

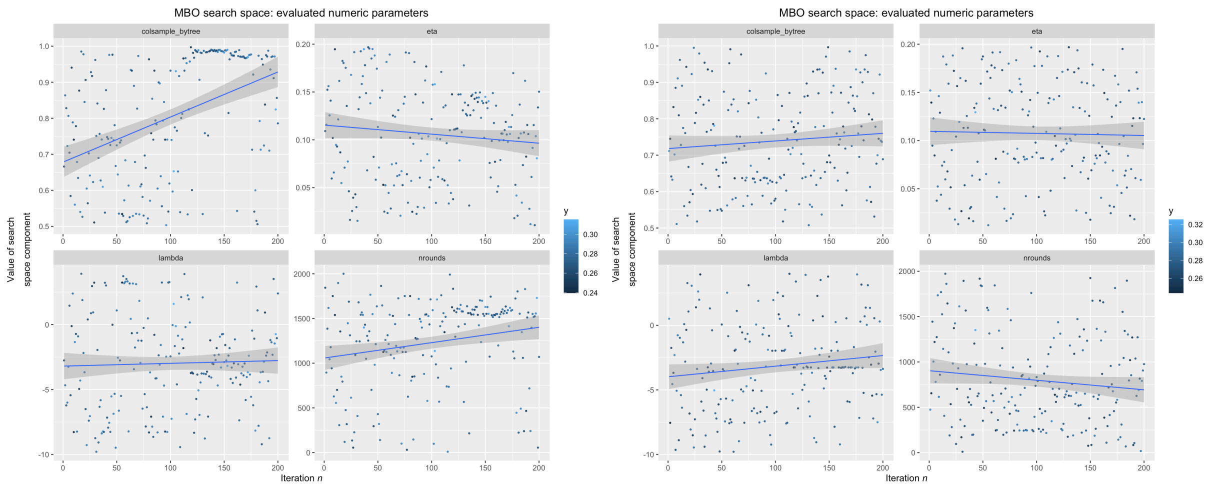


Figure 10: MboPlotSearchSpace: model1, model2. The left (right) plot shows the values of the different search space components (y-axis) that have been evaluated by the optimizer over the number of iterations (x-axis). The color of the points correspond to the value of the objective function. The line shows a linear model fitted on the values of the search space components (dependent variable) over the iterations (independent variable).[44]

plot type we can see the same patterns as identified before, which depicts a high number of exploitation points of *colsample_bytree* for model1 around one, leading to a positive linear dependency over the number of iterations (marked by the regression line of the linear

[44] Own illustration based on R. Example provided in submission-repository.

model). Besides that, we see that most of these points have a low (good) y-value which explains the behavior of the optimizer with a higher degree of exploitation, concentrating on this part of the domain. We also note that the last of the bulk of points sampled around a value of one show a more worse y-value (light blue color), thats why the optimizer begins exploration afterwards again. In general the sampled points for model2 are broader spread over the domains of the single search space components. A modification for this plot type is possible in two ways. We can exclude the infromation on the objective function y and we can also exclude the points from the initial design.

By now we gained insights into the general optimization progress and the search space but we still have limited knowledge on the interrelation (dependencies) among the search space components. We will thus take a look at the plot of the class *MboPlotDependencies* which is provided in figure 11 for model1. In this plot we can check how single features are

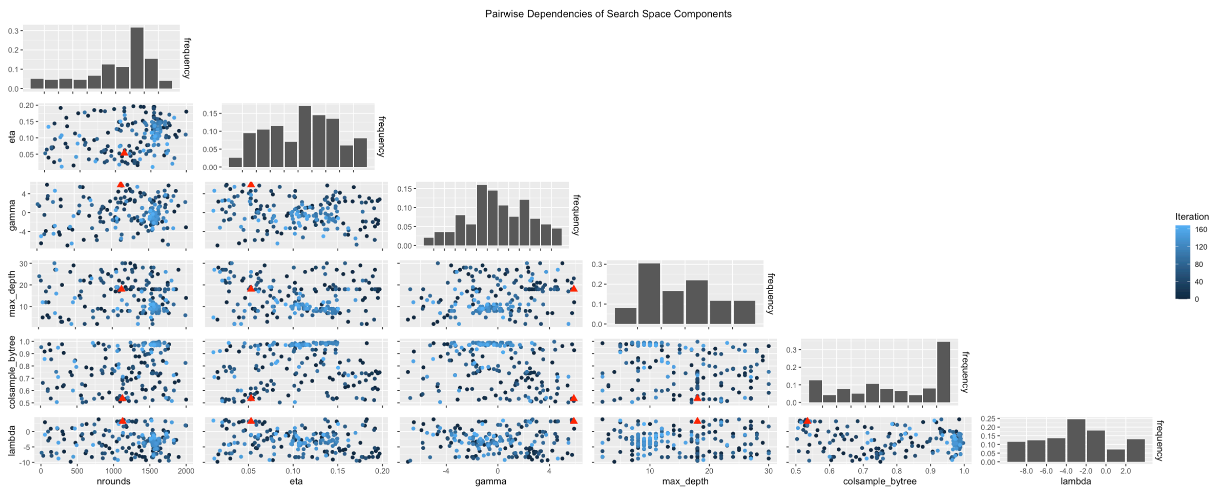


Figure 11: MboPlotDependencies: model1. The plot matrix shows the histogram of the single search space components on the diagonal.The lower triangle shows the pairwise scatter plots of the search space components *nrounds, colsample_bytree, lambda, eta, gamma, max_depth.* The red triangle marks the combination of the two search space components under review which leads to the minimum value of the objective y. The color corresponds to the iteration in which the points were sampled.[45]

related to each other and which combinations the optimizer searches in later iterations. For *nrounds* we can see clear bulks of points around a value of 1,500 with all other features. This may justify our intention to limit the lower boundary of the domain for *nrounds* to

[45] Own illustration based on R. Example provided in submission-repository.

around 500. For *colsample_bytree* our intention from figure 9 was to shift the domain upwards, but we can now see that the minimum y is found at the lower boundary of the domain, thus we might just widen the range of *colsample_bytree*, so it has a higher upper boundary. Another pattern which could be detected in this plot type (which is not present in the chosen example) is a high correlation between two features.[46] In such a case, we could choose the value of one feature depending on the value of the other.

In the last step on the overall run inspection, we take a look at the distance between the search space components. Figure 12 shows the plot of the class *MboPlotDistToNeighbor* which plots the Gower distance against the number of iterations. We use the Gower distance to consider discrete parameters of the search space too.[47] From the plot type of *MboPlotDistToNeighbor* we can take several judgements. First, we can evaluate the size

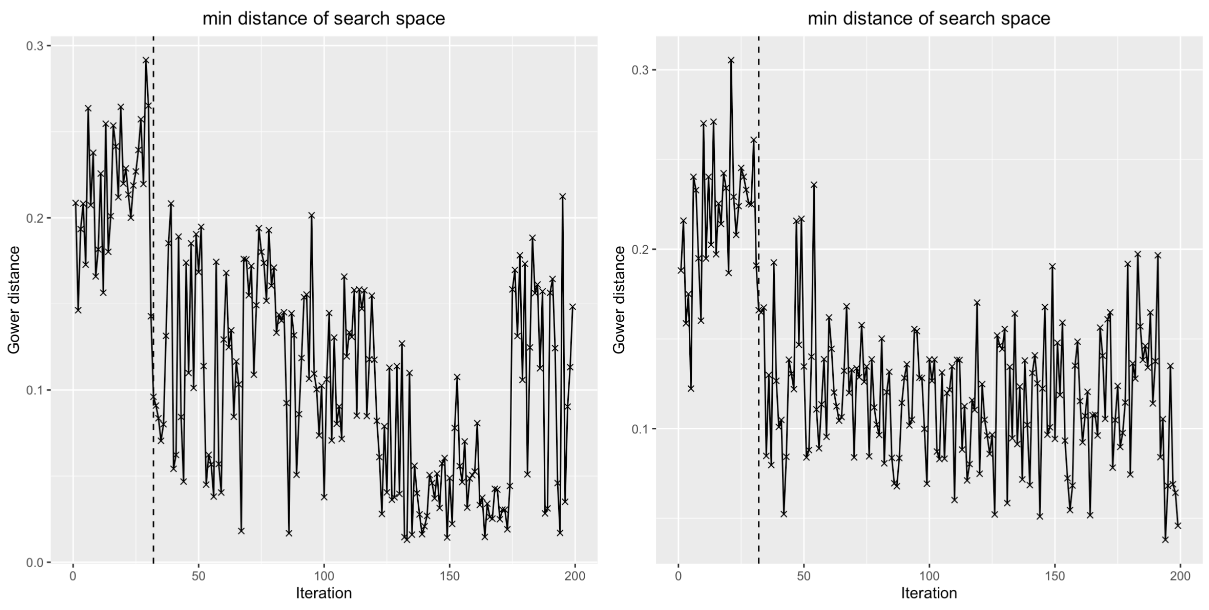


Figure 12: MboPlotDistToNeighbor: model1, model2. The left (right) plot shows the minimum Gower distance between the search space components that have been evaluated by the optimizer over the number of iterations. The vertical line separates the points from the initial design.[48]

of the design. If the Gower distance does not drop significantly after the initial design we might choose a larger number of initial design points. For the example shown in

[46] This correlation always needs to be seen as a correlation which is conditioned on the target.

[47] For an explanation on the formula of the Gower distance please see the description section of *MboPlotDistToNeighbor* in the tab *Exploration vs. Exploitation* in the Shiny application.

[48] Own illustration based on R. Example provided in submission-repository.

figure 12, both models seem to have a sufficient size of the initial design. Besides that we can see the tradeoff between exploration and exploitation of the optimizer. For model1 we see that the Gower distance drops around iteration 130 where the optimizer exploits heavily. These are exactly the iterations which we have highlighted in Figure 9, where the optimizer searched frequently at around one for the feature *colsample_bytree*. For model2 the Gower distance varies in, more or less, the same bandwidth, which shows the character of exploration.

We now took a look at the overall run section, where we can identify general patterns and possible mis-specifications of the optimization run. Now we take a look at the diagnostic section, where we can inspect single iterations of the SMBO run in more detail. We decide to inspect iteration 123 in more details within the subsequent paragraphs.

The first class, *MboPlotRuntime*, plots the time spend during the SMBO run. The associated plot is shown in figure 13. The left plot shows the execution time, i.e. the time spend executing the objective function $f(\mathbf{x})$ which has been passed to the optimizer. The right plot shows the training and proposal time. The training time is the time spend

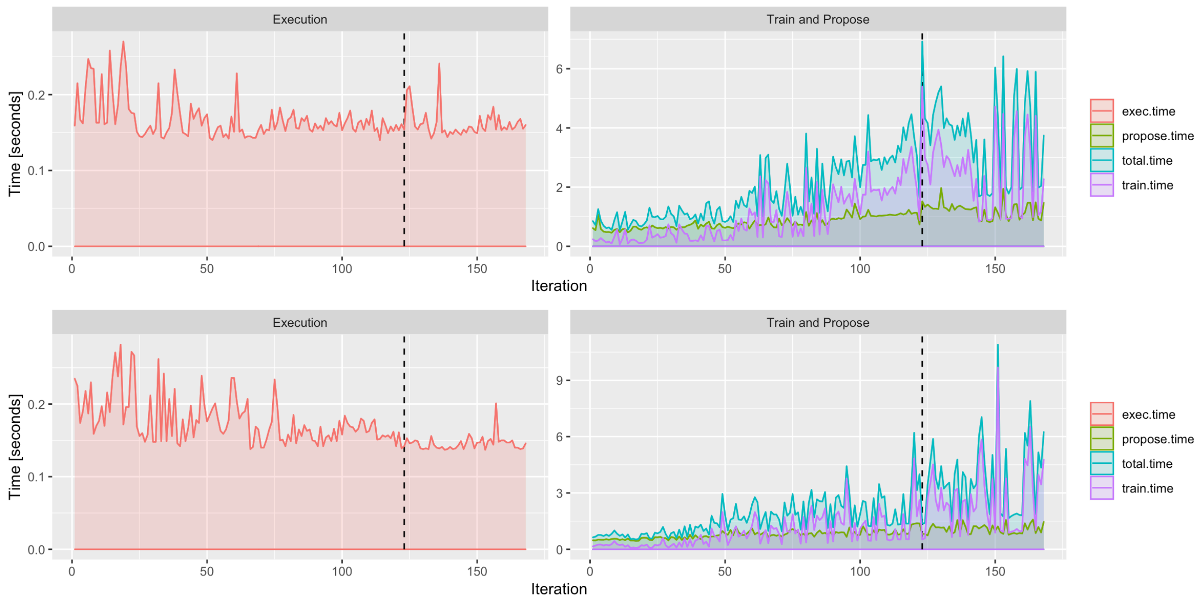


Figure 13: MboPlotRuntime: model1, model2. The upper (lower) plot shows the runtime of the overall mlrMBO run for model1 (model2). The vertical line marks iteration 123.[49]

to train the surrogate model (which proposes the new points). The proposal time is the

[49] Own illustration based on R. Example provided in submission-repository.

time spend on the infill optimization, to propose a new point (given the trained surrogate). This plot type assists the user as it can reveal patterns of an inadequate split of time, spend for surrogate fitting (train time) and the proposal of new points (propose time). Using a GP as surrogate we are sometimes faced with very expensive fitting, especially in later iterations. If such an inadequacy of fitting time, compared to the proposal time of new points, escalates too much we might think of choosing a better suited surrogate. Figure 13 shows quite a high train time in iteration 123 for model1. For model2, which explores more often, there is only one major peak in train time. Keeping these information in mind we now analyze the model fit in more detail.

MboPlotFit offers two kinds of visualizations, the R-squared of the model and the comparison of the predicted target $\hat{y}$ with the true target y. Figure 14 shows both plots for model1 and model2. We see that the in-sample R-squared increases over the iterations

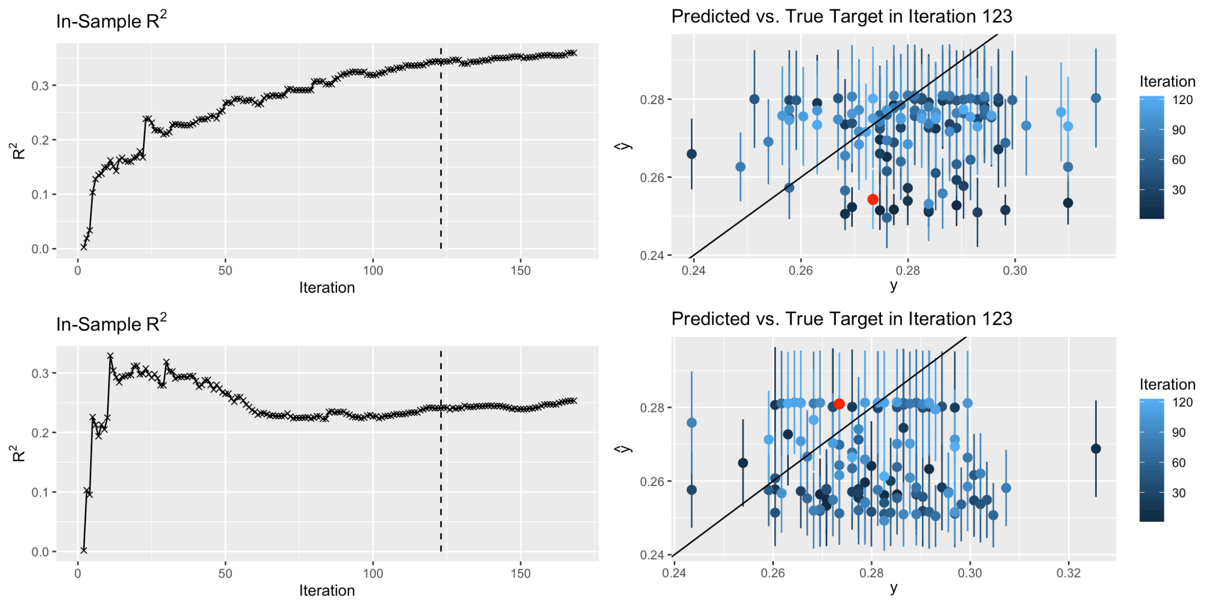


Figure 14: MboPlotFit: model1, model2. The upper (lower) plot shows the fit of the mlrMBO for model1 (model2). The left plots show the in-sample R-squared. The vertical line marks iteration 123. The right plots show the predicted target $\hat{y}$ agains the evaluated target y. The angle bisector marks a 'perfect' prediction of the output y. The color of the points corresponds to the iteration, while the red point marks iteration 123. The vertical lines around the points correspond to the estimated standard deviation of the predicted output.[50]

[50] Own illustration based on R. Example provided in submission-repository.

for model1, while this does not hold for model2. Both models show a poor overall fit, with model1 being superior to model2. From the plots on the right we notice that, in iteration 123 the proposed point $\hat{y}^{(123)}$ (marked by the red point) of model1 is more far away from the true target $y^{(123)}$, than for model2. But we also note that the estimated standard deviation of the proposed point of model1 is smaller than the one of model2, which qualifies our statement on the prediction. To see if the fit may improve, we could choose another surrogate model, e.g. a random forest surrogate, and benchmark both fits against each other. In the next step we will have a more detailed look at the uncertainty in the estimation.

MboPlotEstimationUncertainty helps to visualize the uncertainty in the estimation of new points. In the left plots of figure 15 the uncertainty of the estimation $|\hat{y} - y|$ is depicted for both models. From the right plots in figure 15 we can see that, in general, model1 has

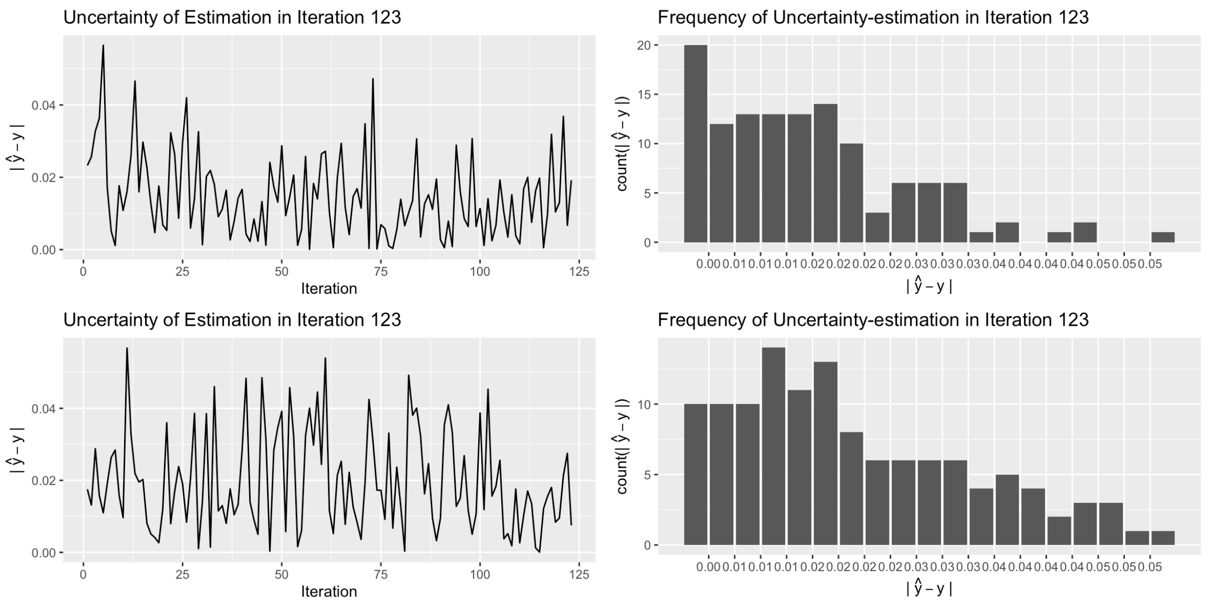


Figure 15: MboPlotEstimationUncertainty: model1, model2. The upper (lower) plot shows the estimation uncertainty for model1 (model2). The left plots show the absolute difference between the estimated output and the true output (i.e. the uncertainty) of the estimation up to iteration 123. The right plots show the frequency of the absolute uncertainty of the 123 iterations.[51]

a smaller estimation uncertainty than model2. For model1 the frequency of the absolute deviations $|\hat{y} - y|$ is highest in the left class with a count of around 20, while for model

[51] Own illustration based on R. Example provided in submission-repository.

2 the highest frequency is around a absolute deviation of 0.02. model1, which exploits more often, only shows a few larger absolute deviations $|\hat{y} - y|$, while for model1 the bar chart flattens out to the right, showing more frequent uncertain estimations. We note that the uncertainty is, in general, decreasing with higher iterations for model1, which is in line with the information on the R-squared from *MboPlotFit* (figure 14). The surrogate of model1 seems to improve over the iterations, even though it found the minimum cumulative value of the objective function already in iteration 25, as we have seen in figure 8. Thus it might hold that an increase of the number of iterations can decrease the cumulative minimum value of model1.

To get an intention how the search space components influence the surrogate model, we will now take a look at *MboPlotOptPath*. In general, the class plots the surrogate model in dependence of the search space. For higher dimensional search spaces the class plots a PDP of the surrogate model with regards to the chosen search space component. For an iteration i, the marginal effect of a specified search space component, on the predicted outcome $\hat{y}^{(i)}$ of the surrogate model, is computed. Figure 16 shows the PDPs for model1

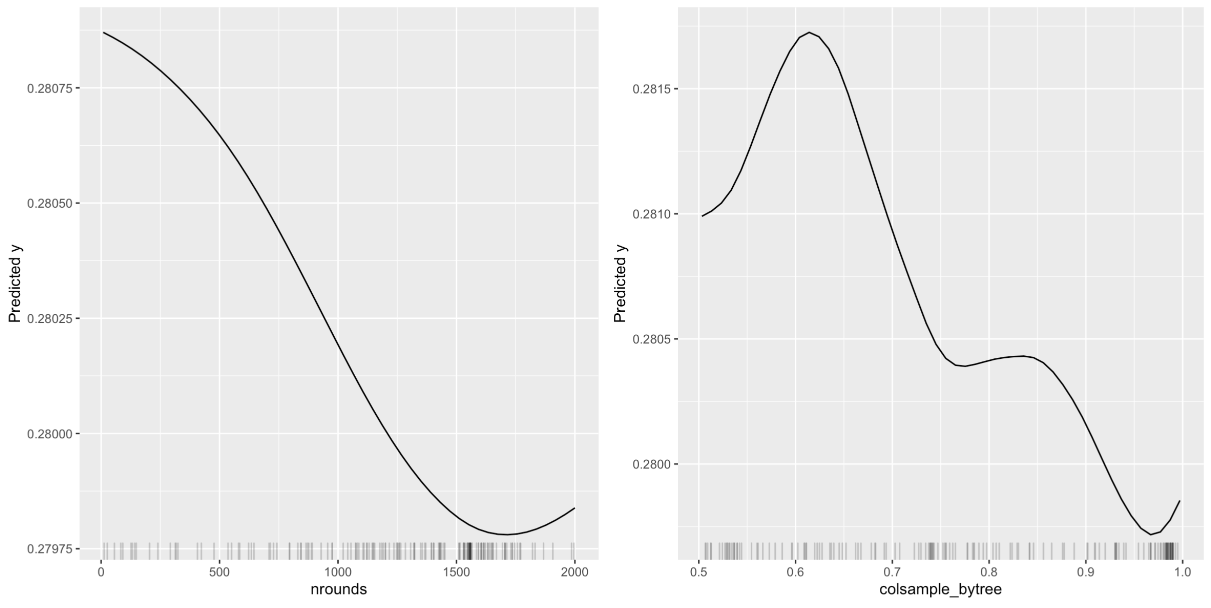


Figure 16: MboPlotOptPath: model1. The left (right) plot shows the PDP of the surrogate model with respect to the search space component *nrounds* (*colsample_bytree*) at iteration 123.[52]

[52] Own illustration based on R. Example provided in submission-repository.

for the features *nrounds* and *colsample_bytree*. In the left plot we see that, for iteration 123, if *nrounds* increases the predicted outcome $\hat{y}^{(123)}$ decreases, given all other search space components. The interpretation of the PDP for *colsample_bytree* follows equivalent.

To summarize our findings we have seen that both models find their minimum cumulative value in an early iteration. We have analyzed this behavior further in the diagnostic section and found model1 to improve its surrogate fit over the iterations, while this does not hold for model2. This is also confirmed by the uncertainty of the estimated output, where model1 shows more lower absolute deviations $|\hat{y} - y|$ than model2 does. Besides that we found some anomalies in the search space definition, which clearly identified the transformation on the random sampled design. We also found a probable mis-specification of the domain, where the optimizer searches frequently at the upper boundary for *colsample_bytree*. This seems to affect model1, which exploits frequently, more than model2. We found that in general the distance between the search space components is closer when we choose a model which exploits more often. The exploitation brings the side effect that the runtime for the training of the GP surrogate shows peaks, while this is rarely the case in a model which explores more often. All in all the different visualizations gave us a variety of insights into the two analyzed SMBO runs that enhanced our general understanding and led to ideas of possible improvements.

4 Outlook and Further Improvements

To conclude this work, we give a short outlook on the future context of *VisBayesOpt* and provide some further improvements to enhance the usability of the package.

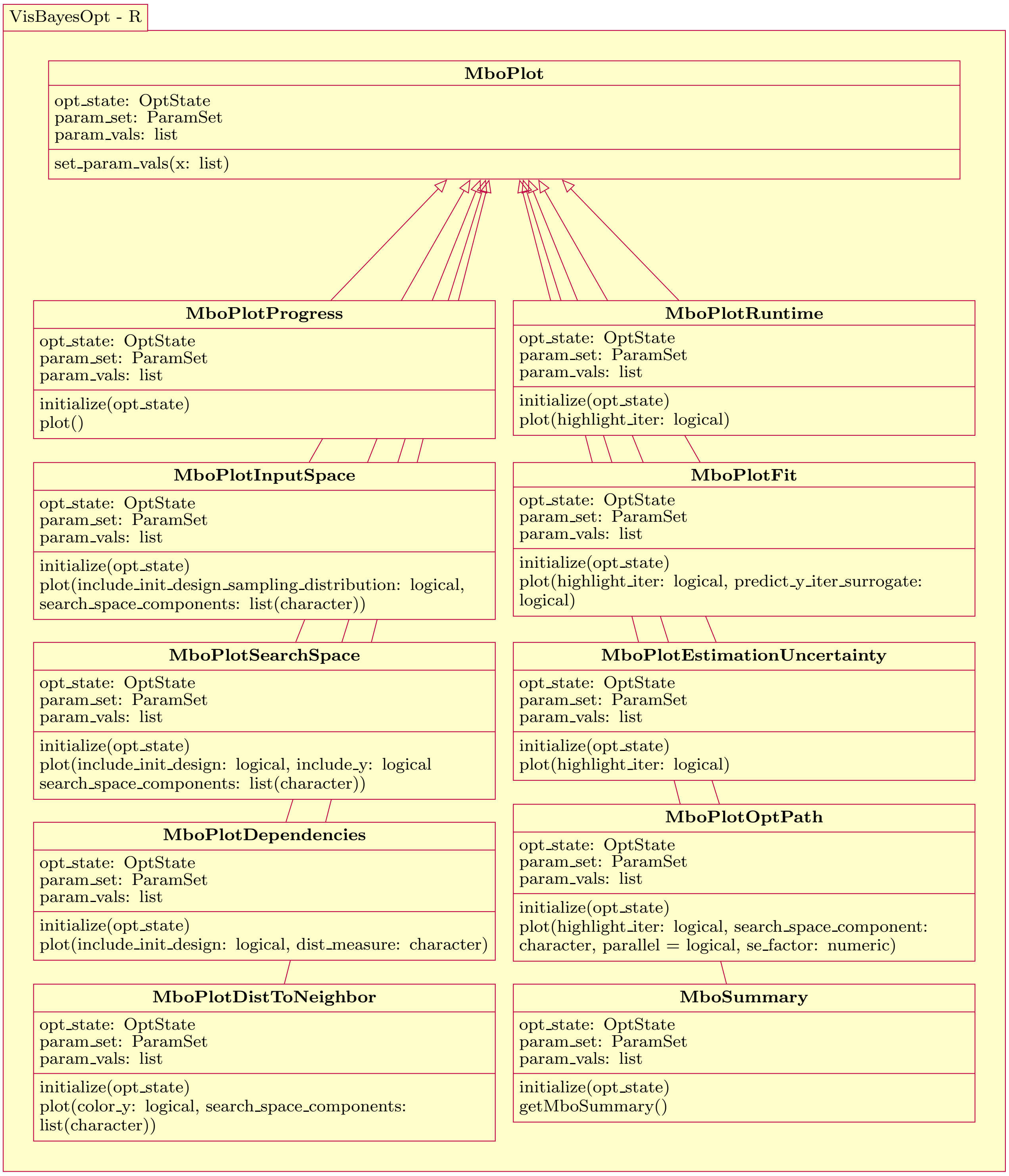
One major challenge in the future is the migration of *VisBayesOpt* to *mlr3*, which is currently under development. To balance the tradeoff between cost and benefit, an adaption to *mlr3* might be justified if *VisBayesOpt* finds sound interest among the users of *mlr2*. Potential improvements may be identified once a broader user group analyzes their *mlrMBO* runs using *VisBayesOpt*. This will also show the degree of the limitations outlined in table 1 (i.e. no mixed-search spaces, no multi-point proposal).

For further improvements of the package itself, we refer to the limitations outlined in table 1. The first step should be to expand the package to handle mixed-space objective functions too. The additional value for visualizing SMBO runs with multi-point-proposals might not exceed the complexity of the implementation. From the technical perspective the package could be enhanced by implementing a full set of test (e.g. by the *testthat* package) to check the functionality of the single functions easily.

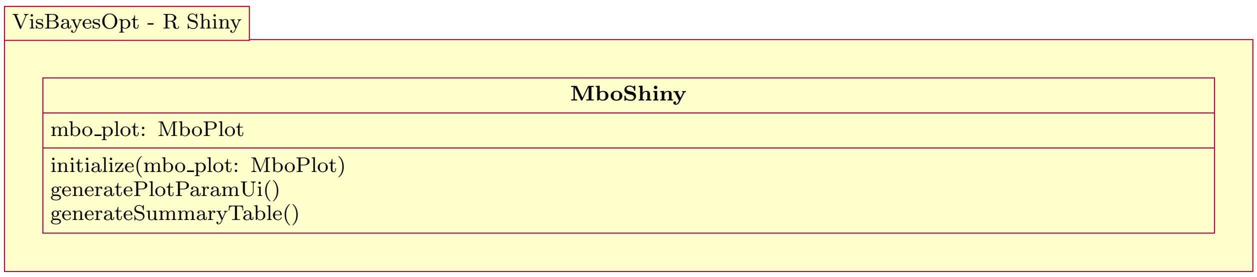
All in all we hope that *VisBayesOpt* attracts a broad group of users from different professions, enhances their insights into SMBO and facilitates their work on a day-to-day basis.

Appendices

Appendix 1: UML diagram of classes in *VisBayesOpt*[53]



[53] Own illustration based on package *VisBayesOpt*. Note that helper functions are not part of the UML diagram.


VisBayesOpt - R Shiny
MboShiny
mbo_plot: MboPlot
initialize(mbo_plot: MboPlot)
generatePlotParamUi()
generateSummaryTable()

References

[1] M. Abramowitz and I. A. Stegun. Handbook of mathematical functions dover publications. *New York*, page 361, 1965.

[2] F. Archetti and A. Candelieri. *Bayesian Optimization and Data Science*. Springer, 2019.

[3] C. Audet and W. Hare. Derivative-free and blackbox optimization. 2017.

[4] A. Baheri, P. Ramaprabhu, and C. Vermillion. Iterative 3d layout optimization and parametric trade study for a reconfigurable ocean current turbine array using bayesian optimization. *Renewable energy*, 127:1052–1063, 2018.

[5] B. Bischl, J. Richter, J. Bossek, D. Horn, M. Lang, and T. Janek. Cran package 'mlr3viz'. `https://cloud.r-project.org/web/packages/mlrMBO/mlrMBO.pdf`, 2020.

[6] B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang. mlrmbo: A modular framework for model-based optimization of expensive black-box functions. *arXiv preprint arXiv:1703.03373*, 2017.

[7] W. Chang. Cran package 'R6'. `https://cloud.r-project.org/web/packages/R6/R6.pdf`, 2019.

[8] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[9] J. Deshmukh, M. Horvat, X. Jin, R. Majumdar, and V. S. Prabhu. Testing cyber-physical systems through bayesian optimization. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–18, 2017.

[10] C. T. et al. Cran package 'xgboost'. `https://cran.r-project.org/web/packages/xgboost/xgboost.pdf`, 2020.

[11] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration (extended version). *Technical Report TR-2010–10, University of British Columbia, Computer Science, Tech. Rep.*, 2010.

[12] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[13] M. Kuhn. Cran package 'tune'. `https://cran.rstudio.com/web/packages/tune/tune.pdf`, 2020.

[14] F. Leisch and E. Dimitriadou. Cran package 'mlbench'. `https://cran.r-project.org/web/packages/mlbench/mlbench.pdf`, 2012.

[15] C. Molnar. *Interpretable Machine Learning.* Lulu. com, 2020.

[16] C. E. Rasmussen and C. Williams. Gaussian processes for machine learning, vol. 1. *MIT press*, 39:40–43, 2006.

[17] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.