Gradient Boosting in Automatic Machine Learning: Feature Selection and Hyperparameter Optimization

Janek Thomas

München 2019

Gradient Boosting in Automatic Machine Learning: Feature Selection and Hyperparameter Optimization

Janek Thomas

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität München

eingereicht von
Janek Thomas
am 21.02.2019

Erstgutachter: Prof. Dr. Bernd Bischl
Zweitgutachter: Prof. Dr. Andreas Mayr
Drittgutachter: Prof. Dr. Eyke Hüllermeier
Tag der Mündlichen Prüfung: 09.04.2019

Danksagungen

Diese Doktorarbeit wäre ohne die Hilfe, Unterstützung und Anleitung vieler Menschen nicht möglich gewesen! Insbesondere möchte ich mich herzlich bedanken bei . . .

. . . *meinem Doktorvater Prof. Dr. Bernd Bischl,*

. . . *Prof. Dr. Andreas Mayr und Prof. Dr. Eyke Hüllermeier als Gutachter dieser Arbeit,*

. . . *allen meinen Kollegen in der Arbeitsgruppe CompStat an der LMU. Besonders Giuseppe Casalicchio, Daniel Schalk, Xudong Sun und Stefan Coors, mit denen ich ein Büro teilte,*

. . . *dem LMU Mentoring-Programm für finanzielle und akademische Unterstützung,*

. . . *Microsoft und H2O.ai für zwei großartige Praktika,*

. . . *und schließlich natürlich bei meiner Familie und Freunden, die mich immer unterstützt haben.*

Zusammenfassung

Das Ziel des automatischen maschinellen Lernens (AutoML) ist es, alle Aspekte der Modellwahl in prädiktiver Modellierung zu automatisieren. Diese Arbeit beschäftigt sich mit Gradienten Boosting im Kontext von AutoML mit einem Fokus auf Gradient Tree Boosting und komponentenweisem Boosting. Beide Techniken haben eine gemeinsame Methodik, aber ihre Zielsetzung ist unterschiedlich. Während Gradient Tree Boosting im maschinellen Lernen als leistungsfähiger Vorhersagealgorithmus weit verbreitet ist, wurde komponentenweises Boosting im Rahmen der Modellierung hochdimensionaler Daten entwickelt. Erweiterungen des komponentenweisen Boostings auf multidimensionale Vorhersagefunktionen werden in dieser Arbeit ebenfalls untersucht. Die Herausforderung der Hyperparameteroptimierung wird mit Fokus auf Bayesianische Optimierung und effiziente Stopping-Strategien diskutiert. Ein groß angelegter Benchmark über Hyperparameter verschiedener Lernalgorithmen, zeigt den kritischen Einfluss von Hyperparameter Konfigurationen auf die Qualität der Modelle. Diese Daten können als Grundlage für neue AutoML- und Meta-Lernansätze verwendet werden. Darüber hinaus werden fortgeschrittene Strategien zur Variablenselektion zusammengefasst und eine neue Methode auf Basis von permutierten Variablen vorgestellt. Schließlich wird ein AutoML-Ansatz vorgeschlagen, der auf den Ergebnissen und Best Practices für die Variablenselektion und Hyperparameteroptimierung basiert. Ziel ist es AutoML zu vereinfachen und zu stabilisieren sowie eine hohe Vorhersagegenauigkeit zu gewährleisten. Dieser Ansatz wird mit AutoML-Methoden, die wesentlich komplexere Suchräume und Ensembling Techniken besitzen, verglichen.

Vier Softwarepakete für die statistische Programmiersprache R sind Teil dieser Arbeit, die neu entwickelt oder erweitert wurden: *mlrMBO*: Ein generisches Paket für die Bayesianische Optimierung; *autoxgboost*: Ein AutoML System, das sich vollständig auf Gradient Tree Boosting fokusiert; *compboost*: Ein modulares, in C++ geschriebenes Framework für komponentenweises Boosting; *gamboostLSS*: Ein Framework für komponentenweises Boosting additiver Modelle für Location, Scale und Shape.

Summary

The goal of automatic machine learning (AutoML) is to automate all aspects of model selection in (supervised) predictive modeling. This thesis deals with gradient boosting techniques in the context of AutoML with a focus on gradient tree boosting and component-wise gradient boosting. Both techniques have a common methodology, but their goal is quite different. While gradient tree boosting is widely used in machine learning as a powerful prediction algorithm, component-wise gradient boosting strength is in feature selection and modeling of high-dimensional data. Extensions of component-wise gradient boosting to multidimensional prediction functions are considered as well. Focusing on Bayesian optimization and efficient early stopping strategies the challenge of hyperparameter optimization for these algorithms is discussed. Difficulty in the optimization of these algorithms is shown by a large scale random search on hyperparameters for machine learning algorithms, that can build the foundation of new AutoML and meta-learning approaches. Furthermore, advanced feature selection strategies are summarized and a new method based on shadow features is introduced. Finally, an AutoML approach based on the results and best practices for feature selection and hyperparameter optimization is proposed, with the goal of simplifying and stabilizing AutoML while maintaining high prediction accuracy. This is compared to AutoML approaches using much more complex search spaces and ensembling techniques.

Four software packages for the statistical programming language R have been newly developed or extended as a part of this thesis: *mlrMBO*: A general framework for Bayesian optimization; *autoxgboost*: An automatic machine learning framework that heavily utilizes gradient tree boosting; *compboost*: A modular framework for component-wise boosting written in C++; *gamboostLSS*: A framework for component-wise boosting for generalized additive models for location scale and shape.

Contents

Contributions of the Thesis

This cumulative PhD thesis consists of seven publications listed below:

1. Thomas J., Hepp T., Mayr A., Bischl B. (2017) Probing for Sparse and Fast Variable Selection with Model-Based Boosting *Computational and Mathematical Methods in Medicine*
2. Bischl B., Richter J., Bossek J., Horn D., Thomas J., Lang M. (2017) mlrMBO: A modular framework for model-based optimization of expensive black-box functions *arXiv preprint*, 1703.03373
3. Kotthaus H., Richter J., Lang A., Thomas J., Bischl B., Marwedel J., Rahneführer J., Lang M. (2017) RAMBO: Resource-Aware Model-Based Optimization with Scheduling for Heterogeneous Runtimes and a Comparison with Asynchronous Model-Based Optimization International Conference on Learning and Intelligent Optimization, 180-195
4. Thomas J., Mayr A., Bischl B., Schmid M., Smith A., Hofner B. (2018) Gradient boosting for distributional regression: faster tuning and improved variable selection via noncyclical updates. *Statistics and Computing*, 28:673–687.
5. Kühn D., Probst P., Thomas J., Bischl B. (2018) Automatic Exploration of Machine Learning Experiments on OpenML *arXiv preprint*, 1806.10961
6. Thomas J., Coors S., Bischl B. (2018) Automatic Gradient Boosting. *International Workshop on Automatic Machine Learning at ICML*
7. Schalk D., Thomas J., Bischl B. (2018) compboost: Modular Framework for Component-Wise Boosting. *The Journal of Open Source Software*, 00894

They are attached to the thesis as Chapters 4 to 10. The individual contributions of the authors are described in the respective chapters.

1. Introduction

> Society is built on interfaces. You take a complex thing, put it inside a sturdy box, and put some simple buttons on the box so that people can use the thing inside. The box makes it easier to use and prevents people from breaking it. For example, you can take the machinery of a clock, put it in a box, and put two hands on the outside along with a knob for winding it. Take all the machinery of a car, hide it behind a dashboard, and give people two pedals and wheel. Take all the circuits of a computer, put them in a box, and give people a monitor and a keyboard.
>
> (The Interface Series)

More and more areas of daily life are influenced, enhanced or even controlled by automatic decision-making machine learning algorithms. Machines recommend routes based on current traffic information, suggest music to listen to and which restaurant we are likely to enjoy most for lunch. The above described services can be convenient but also easily ignored if not desired. This is no longer possible when an automatic decision-making process has a greater and more lasting impact on our lives. When machines pick the interest rate of a loan or decide whether a financial transaction is fraudulent, mistakes can be catastrophic. For this reason, it is crucial that the machine learning models that govern the decision-making process are optimally selected and configured.

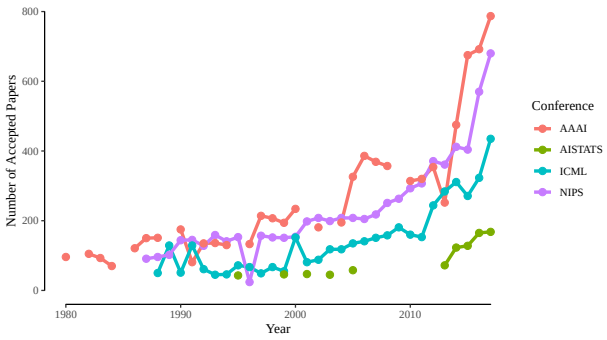

Figure 1.1.: Number of accepted papers in four of the most popular machine learning conferences. Conference on Neural Information Processing Systems (NIPS), Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence (AAAI), International Conference on Machine Learning (ICML) and International Conference on Artificial Intelligence and Statistics (AISTATS). Source: `http://dblp.org`

In recent years, the number of machine learning algorithms has increased dramatically. It is difficult to quantify the number of available algorithms, but if submissions to machine learning conferences (Figure 1.1) are an indicator, a sharp increase is visible. The increasing number of methods makes it more difficult for practitioners and researchers in the field of machine learning to keep up with new developments and to

have a sound practical knowledge of these algorithms. However, this knowledge can be important to choose the correct model for a particular problem.

Although these developments are assessed positively by most researchers, they demonstrate the need for automatic and efficient model selection in machine learning. Otherwise, it will be difficult to meet the growing demand for machine learning specialists. The optimal search and selection of hyperparameters for machine learning is a well-studied field of research that has led to more complex and highly configurable ML algorithms. In numerous recent publications, these techniques have been used as a basis to include the model choice itself as a hyperparameter for optimization. With the method itself as a hyperparameter and inclusion of preprocessing steps, the tuning space becomes a lot more complex due to its size and hierarchical structure. An example of such a space is shown in Figure 1.2.

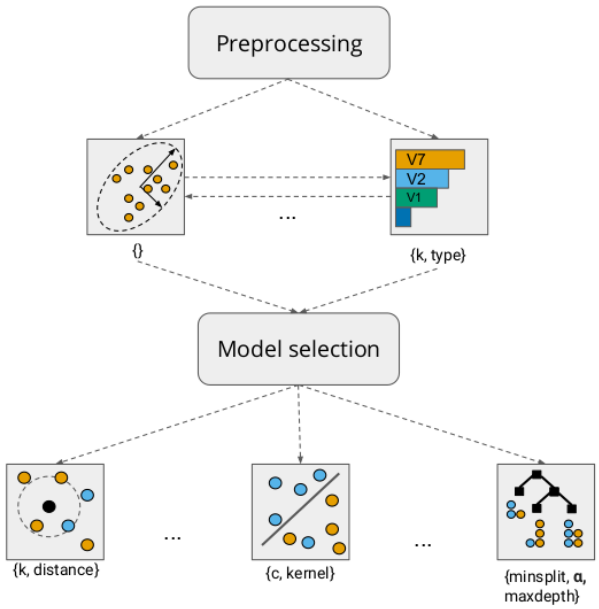

Figure 1.2.: Example of a machine learning pipeline. The preprocessing step is composed of one or multiple transformations (exemplary shown are principal component analysis and feature filtering). Subsequent is the model selection step in which a single model is selected (exemplary shown are k-nearest neighbors, a linear support vector machine and a classification tree). The pipeline is defined as a directed acyclic graph (DAG), assuming that no preprocessing step can be visited twice. Each node in the DAG can add numerical and categorical hyperparameters to the optimization space.

It is difficult to pinpoint the beginning of automatic machine learning. The term *model selection* has been around for a long time, but can refer to different steps in the model building process, like hyperparameter optimization or feature selection (Escalante et al., 2009). One of the first publications that jointly optimizes preprocessing steps, feature selection and machine learning model choice is by Escalante et al. (2009), who call this process *full model selection* and propose to use *Particle Swarm Optimization* to tune over this joint space. Thornton et al. (2013) call this process *Combined Algorithm Selection and Hyperparameter optimization* (CASH). In general, this problem can be referred to as *machine learning pipeline configuration*, since the optimization space is describable as a directed acyclic graph (DAG) of discrete preprocessing, modeling and postprocessing steps. An example of such a graph can be seen in Figure 1.2. Each node in these graphs can be associated with numeric or discrete hyperparameters. As a consequence, optimizing over such a space of pipelines becomes a challenging problem because the configuration space takes on a hierarchical structure and most standard black-box optimizers assume a flat, often even fully numeric, configuration space. More and more frameworks and methodologies for this problem have been introduced in recent years, using *Bayesian optimization* (Feurer et al., 2015), *genetic algorithms* (Olson et al., 2016) or *planning techniques* (Mohr et al., 2018).

With the success of (deep) neural networks in the past years and their high difficulty in being configured optimally, AutoML techniques have been applied to the problem of finding an optimal networks architecture, i.e, number, size and type of layers. This problem is usually referred to as *Neural Architecture Search* (Zoph and Le, 2016; Zoph et al., 2017). In this thesis architecture search will not be considered in detail, instead the focus will be on classical machine learning techniques for tabular data.

Following, a short overview of current AutoML systems is given and in Table 1.1 their configuration spaces and optimization strategies are briefly summarized.

auto-sklearn (Feurer et al., 2015) is an AutoML framework developed in Python. It is based on the machine learning framework `Scikit-learn` (Pedregosa et al., 2011) and the Bayesian optimization framework `SMAC` (Hutter et al., 2011). It uses meta-learning based on `OpenML` data (Vanschoren et al., 2014) to initialize the optimization process and uses stacking to create powerful ensembles.

Auto-WEKA (Thornton et al., 2013; Kotthoff et al., 2017) is implemented in Java and applies Bayesian optimization to tune over a space of classifiers and their hyperparameters implemented in the WEKA machine learning library (Holmes et al., 1994; Hall et al., 2009). It has stacking of base algorithms as part of its search space. An interface to python exists.

H2O AutoML (H2O.ai, 2019a) is based on the `H2O` machine learning framework H2O.ai (2019b) implemented in Java. It performs a random search and ensembles models implemented in `H2O`. Interfaces to Python and R are available.

hyperopt-sklearn (Komer et al., 2014) also uses `Scikit-learn`, but uses `hyperopt` (Bergstra et al., 2013) as its optimizer to offer random search, simulated annealing and an optimizer based on trees of Parzen estimators (Bergstra et al., 2011) to search the configuration space.

TPOT (Olson et al., 2016) is again based on machine learning algorithms from `Scikit-learn`, but uses genetic algorithms to sample, modify and combine machine learning pipelines.

ML-Plan (Mohr et al., 2018) is implemented in Java and based on hierarchical task network planning (Erol et al., 1994). It can be applied to `WEKA` as well as `Scikit-learn` machine learning algorithms.

Furthermore there are multiple approaches that discretize the search space and use some form of collaborative filtering (Fusi et al., 2018; Yang et al., 2018; Sun-Hosoya, Guyon, and Sebag, Sun-Hosoya et al.). The general concept of ensembling or stacking (Caruana et al., 2004) is quite

	Machine learning framework				
Name	Scikit-learn	WEKA	H2O	xgboost	Optimization method
auto-sklearn	✓			✓	Bayesian optimization
auto-WEKA		✓			Bayesian optimization
H2O AutoML			✓	✓	Random search
hyperopt-sklearn	✓			✓	*Various*
TPOT	✓			✓	Genetic algorithms
ML-Plan	✓	✓		✓	Hierarchical task networks

Table 1.1.: Overview of AutoML systems.

important in AutoML and is used by a large number of the frameworks introduced above. Some AutoML approaches, such as Wistuba et al. (2017), produce a massive amount of models that are stacked in multiple layers and achieve very strong performance.

Many more AutoML systems are rapidly developed and published, making a thorough comparison and benchmark between these systems difficult. One approach which is currently still in development is called automlbenchmark*. It is based on Docker containers and OpenML for reproducibility and extendability.

Since 2015 three AutoML challenges have been organized by *automl.chalearn*†. The first two challenges have been won by modified versions of autosklearn (Guyon et al., 2016; Feurer et al., 2018). Interestingly, for the third challenge, AutoML systems relying exclusively on a single machine learning algorithm were used by both the first and second place teams. The first place AutoGBT (Wilson et al., 2018) and second place "A Boosting Tree Based AutoML System for High Cardinality Streaming Data Classification with Concept Drift", used exclusively gradient tree boosting. The success of these approaches in the third challenge could have various reasons, the data was much larger than in the previous competitions while the provided computing resources were not scaled proportionally. The data also exhibited concept drift, i.e., a time-dependent change in data distribution, which made automatic adaption by AutoML systems a requirement. However, it is still noteworthy that AutoML systems using only gradient tree boosting appear to be competitive with methods with much larger configuration spaces. Chapter 9 presents an AutoML system that, similar to the first and second place of the third AutoML challenge, is only based on gradient boosting and feature engineering. The importance of gradient boosting in AutoML can also be seen in Table 1.1, in which all frameworks except for auto-WEKA contain xgboost (Chen and Guestrin, 2016), an extremely flexible and efficient implementation of the gradient tree boosting algorithm.

Tree based machine learning methods, such as random forests (Breiman, 2001), extraTrees (Geurts et al., 2006) and gradient tree boosting (Friedman, 2001), have many desirable properties

*https://github.com/openml/automlbenchmark
†http://automl.chalearn.org/

for AutoML approaches. They can often natively deal with missing values, are stable against outliers, can process categorical features with a large number of possible levels and use embedded feature selection. Fernández-Delgado et al. (2014) discuss the question of whether *we need hundreds of classifiers to solve real world classification problems* except the random forests. In this publication, no hyperparameter optimization is performed for the comparison of machine learning methods. This is questionable in respect to the random forest, which is a very stable algorithm regarding its hyperparameters and requires little tuning (Probst and Boulesteix, 2018). Nevertheless, gradient tree boosting with optimized hyperparameters often surpasses random forests, as can be seen in its very strong performance in machine learning competitions (Chen and Guestrin, 2016). Since gradient tree boosting is such an important algorithm, its fundamental principles and properties for automatic machine learning are discussed in this thesis.

In addition, the gradient boosting algorithm can be easily adjusted to boost generalized linear or additive model (Hothorn et al., 2010), which is then called *model-based* or *component-wise* gradient boosting. Component-wise gradient boosting has strong feature selection characteristics (Mayr et al., 2012; Hofner et al., 2015) and feature effects of the model can be easily visualized with the disadvantage that tree-based boosting often produces a stronger predictor.

2. Methodological and General Background

2.1. Gradient Boosting

Boosting is probably one of the most important techniques in statistical- and machine learning. It is based on the *weak learner theorem* by Schapire (1990) which states *"that a model of learnability in which the learner is only required to perform slightly better than guessing is as strong as a model in which the learner's error can be made arbitrarily small"*. The question was initially posed by Kearns and Valiant (1994).

For a given data set $\mathcal{D} = \left\{ \left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(n)}, y^{(n)}\right) \right\}$, sampled i.i.d. from of $\mathcal{X} \times \mathcal{Y}$, a hypothesis is defined as a mapping $h : \mathcal{X} \to \mathcal{Y}$ and the set of all feasible candidate mappings is referred to as *hypothesis class* H. A learner, also called an inducer, takes $\mathcal{D}$ and generates a hypothesis h from H,

$$h : \left\{ \left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(n)}, y^{(n)}\right) \right\} \subset \mathcal{X} \times \mathcal{Y} \to H. \tag{2.1}$$

A weak learner is defined as a learner which only has a slightly better predictive performance than random guessing, e.g., a misclassification rate of slightly less than 0.5 in binary classification (assuming balanced classes). The fundamental idea of boosting is to form a sequential *ensemble* of such learners where each weak learner is applied to the previous model's error. The models built from these weak learners are finally combined into a single model. As these weak learners are the basis of the final model, they are called *base-learners*. The final model is denoted by f to distinguish it from the base-learner models h. Together with *bagging* (Breiman, 1996) and *stacking* (Caruana et al., 2004), boosting is one of the basic approaches of *ensemble methods*. The main difference between boosting and bagging is that in bagging, several slightly modified versions of $\mathcal{D}$ are created, most commonly by bootstrapping. The models are then trained independently on each data set, as opposed to boosting, where the models are trained sequentially. Stacking trains different model types and uses their predictions as new and additional features in a new layer, which creates multiple *stacks* of machine learning models. In addition, strong learning algorithms are considered in a stacking ensemble instead of only weak-learners.

The first boosting algorithm, *AdaBoost* of Freund and Schapire (1997), trains base-learners and iteratively reweighs $\left(x^{(i)}, y^{(i)}\right) \in \mathcal{D}$ based on the errors of previous models and aggregates the

Algorithm 1: Adaboost algorithm (Freund and Schapire, 1997)

Input: Data $\mathcal{D}$, hypothesis space H, number of iterations m_{stop}.

Initialize weights $w^{[1](i)} = \frac{1}{n} \quad \forall i \in \{1, \ldots, n\}$

for $m \in \{1, \ldots, m_{stop}\}$ **do**

$h^{[m]} = \text{fit}(H, \mathcal{D}, w^{[m]})$

$$\epsilon^{[m]} = \frac{\sum_{i=1}^{n} w^{[m](i)} \mathbb{I}(y^{(i)} \neq h^{[m]}(x^{(i)}))}{\sum_{i=1}^{n} w^{[m](i)}}$$

$\beta^{[m]} = \frac{1}{2} \log(\frac{1-\epsilon^{[m]}}{\epsilon^{[m]}})$

$w^{[m+1](i)} = w^{[1](i)} \exp(\beta^{[m]} \mathbb{I}(y^{(i)} \neq h^{[m]}(x^{(i)}))) \quad \forall i \in \{1, \ldots, n\}$

return $f(x) = \sum_{m=1}^{m_{stop}} \beta^{[m]} h^{[m]}(x)$

models as a weighted sum of each model's individual performance. Algorithm 1 formally describes this algorithm.

In the most commonly used version of boosting, base-learners are not trained on reweighed data $\mathcal{D}$, but on $(x^{(i)}, r^{(i)})$, where $r^{(i)}$ is the negative gradient of the loss $L\left(y^{(i)}, f\left(x^{(i)}\right)\right)$, evaluated by the current model at each observation in $\mathcal{D}$, i.e.,

$$r^{[m](i)} = - \left[\frac{\partial L\left(y^{(i)}, f\left(x^{(i)}\right)\right)}{\partial f\left(x^{(i)}\right)} \right]_{f=f^{[m-1]}} . \tag{2.2}$$

These *pseudo residuals* point towards the direction of steepest descent with regard to the risk, in the function space H, where functions are evaluated only at $x^{(1)}, \ldots, x^{(n)}$. This approach is called *gradient boosting* and was initially developed by Friedman (2001). Algorithm 2 describes this general approach.

Algorithm 2: Gradient boosting algorithm (Friedman, 2001)

Input: Data $\mathcal{D}$, hypothesis space H, number of iterations m_{stop}, learning rate ν.

Initialize $f^{[0]} = \arg\min_c \sum_{i=1}^{n} L(y^{(i)}, c)$

for $m \in \{1, \ldots, m_{stop}\}$ **do**

$r^{[m](i)} = - \left[\frac{\partial L\left(y^{(i)}, f\left(x^{(i)}\right)\right)}{\partial f\left(x^{(i)}\right)} \right]_{f=f^{[m-1]}}, \quad \forall i \in \{1, \ldots, n\}$

$h^{[m]} = \arg\min_{h \in H} \sum_{i=1}^{n} (r^{[m](i)} - h(x^{(i)}))^2$

$\beta^{[m]} = \arg\min_{\beta} \sum_{i=1}^{n} L(y^{(i)}, f^{[m-1]}(x^{(i)}) + \beta h^{[m]}(x^{(i)}))$

$f^{[m]} := f^{[m-1]} + \nu \beta^{[m]} h^{[m]}$

return $f^{[m]}$

This is an extremely flexible and versatile algorithm which is able to create models with extremely different properties, depending on the choice of H. It is also a superset of the Adaboost algorithm, since gradient boosting with exponential loss

$$L(y, f(x)) = \exp(-yf(x)), \tag{2.3}$$

is identical to Adaboost. A proof can be found in Friedman et al. (2001). It is more flexible than Adaboost because it works on problems other than binary classification by choosing a suitable loss function, e.g., $L\left(y^{(i)}, f\left(x^{(i)}\right)\right) = (y^{(i)} - f\left(x^{(i)}\right))^2$, for regression. Extensions to multiclass classification are also available and are discussed in Section 2.1.4.

2.1.1. Gradient Tree Boosting

The basic idea is to limit the hypothesis space H to regression trees (Breiman, 1984)

$$H = \left\{ \sum_{t=1}^{T} c_t \mathbb{I}(x \in R_t) \, | c_t \in \mathbb{R}, R_t = \text{box}_m(\mathcal{X}) \quad \forall t = 1, \ldots, T \right\}. \tag{2.4}$$

With boxes R_t, axis parallel to $\mathcal{X}$, such that $\bigcup_{t=1,\ldots,T} R_t = \mathcal{X}$ and $R_t \cap R_{t'} = \emptyset \quad \forall t \neq t'$. A major advantage of trees is that steps 4 and 5 of Algorithm 2 can be performed in a single operation, since trees are piecewise constant models and every leaf can be fitted with regard to an arbitrary loss function.

$$\tilde{c}_t^{[m]} = \underset{c}{\text{arg min}} \sum_{x^{(i)} \in R_t^{[m]}} L(y^{(i)}, f^{[m-1]}(x^{(i)}) + c) \tag{2.5}$$

Trees themselves have many desirable properties:

Training speed: Trees can be trained quite fast. The time required for the initial sorting of all features $1, \ldots, p$ is $\mathcal{O}(pn \log(n))$. The actual split finding for the tree is also typically around $pn \log(n)$ but can take up to $\mathcal{O}(pn^2)$ (Friedman et al., 2001). This can be further accelerated by not iterating over each possible split point, but only quantiles (Chen and Guestrin, 2016).

Missing values: Trees can handle missing values by creating surrogate splits that behave similarly to the split containing missing observations (Breiman, 1984). Alternatively, numeric missing values can be encoded as *out of range*, e.g., twice the largest observed value, which allows the tree to easily split these values in a separate node which contains the missing information. In categorical features missing values can be encoded as a new category. One of the simplest ways is to define a *default split direction*. This direction can be learned

for every node by firstly choosing the optimal split ignoring missing values and then checking if the split quality is higher when the observations with missing values go to the left or to the right child (Chen and Guestrin, 2016).

Outliers and skewness: Since trees only take into account the order of observations, outliers and very skewed feature distributions have no detrimental influence on them. This reduces required preprocessing, e.g., $\log$-scaling skewed features.

Categorical features: Trees can often optimally treat categorical features natively in an efficient way. This is done by ordering the categories based on the average response in regression or for classification by the relative frequency of the positive class. Only splits in this order are considered. This scales linearly with the number of categories, whereas considering all possible partitions grows exponentially. It was originally proposed for regression by Fisher (1958) and for binary classification by Breiman (1984). No proven efficient optimal method is known for multiclass classification.

Gradient tree boosting is very susceptible to overfitting, as the boosting algorithm focuses on smaller subsets of the data in later iterations (Rashmi and Gilad-Bachrach, 2015). Additional regularization by tree depth as well as $L2$ and $L1$ penalty terms are used in most boosting frameworks (Chen and Guestrin, 2016; Ke et al., 2017). These regularizations can easily be added to Equation 2.5 and allow the fitting of terminal regions to be based on the regularized risk. Rashmi and Gilad-Bachrach (2015) suggest additional regularization by using a dropout mechanism (Srivastava et al., 2014) in the boosting process. In each iteration, a random subset of previously trained trees is ignored, when a new tree is added to the ensemble. Finally, the new tree is scaled by the factor of $\frac{|d|}{|d|+1}$, with d being the subset of active trees in this iteration. Such additional regularization techniques allow growing more and deeper trees without overfitting. Extensions to ranking problems (Burges, 2010) as well as the modeling of survival times (Chen et al., 2013) exist. A more efficient use of time and memory in the tree building process can be achieved by not considering every possible split point but only some quantiles of the ordered data (Chen and Guestrin, 2016; Zhang and Wang, 2007). These quantiles can either be recomputed after every split or completely precomputed. As a result, trees will be worse predictors, which can be compensated by using more and deeper trees.

2.1.2. Component-Wise Boosting

In component-wise boosting, the hypothesis space H is limited to additive statistical models, like generalized linear or additive models defined on single features or small sets of features. In simplest case the hypothesis space is restricted to simple linear regression terms, i.e.,

$$H = \{\theta_{0j} + \theta_{1j} x_j | (\theta_{0j}, \theta_{1j}) \in \mathbb{R}^2, j = 1, \ldots, p\}. \tag{2.6}$$

Bühlmann et al. (2007) argue that in this case the line search in step 5 of Algorithm 2 can be ignored. According to Friedman (2001), the line search is obsolete for the quadratic loss. Furthermore, by using a first order Taylor approximation on the risk at $f^{[m-1]}$ for arbitrary loss

Function call	Description
`bols(x)`	Linear effect $\boldsymbol{x}^T\beta$ with intercept (dummy coding for factors)
`bols(x, by = y)`	Linear effect with interaction
`bbs(x)`	Smooth P-spline
`bbs(x, by = z)`	Varying coefficient
`bspatial(x)`	Spatial effect: tensor product P-spline
`bmrf(x)`	Discrete spatial effect: Markov random field
`brandom(x)`	Random intercept
`brandom(x, by = x)`	Random slope

Table 2.1.: Overview of the important base-learners in `mboost` (Hothorn et al., 2018)

functions and without line search, the risk looks very similar to the squared error case up to a factor of $1 - \frac{\nu}{2}$. They conclude that as long as the learning rate ν is sufficiently small, the line search does not have a high influence on the results.

In general more complex additive base-learners, e.g., smooth univariate or multivariate splines, radial basis functions, random effects or Markov random fields can be used.

$$H = \{s(x) | s := \text{additive regression model on single or multiple } x_j, j = 1, \ldots, p\}. \tag{2.7}$$

This enables extremely flexible modeling for interpretable models. An important aspect to keep in mind is the different flexibility of different types of base-learners in component-wise boosting. Suppose that both a linear effect and a smooth p-spline base-learner on the same feature are included in the hypothesis space. There will be an inherent preference to the more flexible p-spline, as it can also approximate a linear effect (Hofner et al., 2011). While this effect can be ignored and only flexible base-learners can be used, it may be desirable to only use smooth effects if they deviate from simple linear effects. Although this can be achieved by feature selection techniques, a more efficient way is to split base-learners of numeric features into three parts: A constant offset, a linear effect without intercept, and a smooth deviation from the linear effect, e.g., by a centered spline. At the end, one needs to ensure that each of these parts has the same flexibility, e.g., by limiting their degrees of freedom. Hofner et al. (2011) propose such a framework for unbiased selection between linear and smooth effects as well as continuous and categorical features. An important implementation of this method is available in the `mboost` package (Hothorn et al., 2018) for the programming language R. It contains a large number of base-learners listed in Table 2.1.

A new implementation for component-wise boosting, called `compboost`, is discussed in Chapter 10. It is implemented in `C++` for speed and improved memory usage, yet provides an interface to R via `Rcpp` (Eddelbuettel and François, 2011).

2.1.3. Parallel Gradient Boosting

With the increased availability of more and more parallel computing capabilities on modern computers, the inherent parallelism of machine learning algorithms becomes more and more important. Gradient boosting is a sequential algorithm, which means parallelizing the algorithm is not trivial. For gradient tree boosting, efficient parallelism can be introduced into the split finding process of the tree fit (Tyree et al., 2011). As the potential split points are precomputed, they can be evaluated in parallel. Such a form of parallelism is implemented in most of the latest efficient gradient boosting implementations, like `xgboost` (Chen and Guestrin, 2016), `lightGBM` (Ke et al., 2017) or `catboost` (Prokhorenkova et al., 2018). For component-wise boosting, parallelism is less studied. A simple but efficient way of doing this would be to parallelize the selection of base-learner in each iteration of the algorithm. Currently, no component-wise boosting framework is implementing such a form of parallelism*.

2.1.4. Gradient Boosting for Multiclass Classification

Another non-trivial extension for gradient boosting is the handling of multi-class classification. In general, the problem can be reduced to several binary classification problems by considering 1-vs-1 or 1-vs-all subproblems. This has a considerable disadvantage of increased complexity and lack of guarantees of an optimal joint predictor (Saberian and Vasconcelos, 2011). A more formal way for gradient tree boosting was introduced by Friedman (2001). In each iteration a model f_k is trained for each class $k = 1, \ldots, g$ and the softmax function

$$\pi_k = \frac{\exp f_k(x)}{\sum_{j=1}^{g} \exp f_j(x)} \tag{2.8}$$

is applied to obtain probability distribution. This model can be trained with a multinomial loss

$$L(y, f(x)) - \sum_{k=1}^{g} \mathbb{I}(y = k) \log \pi_k. \tag{2.9}$$

Growing trees with this loss create the problem that the optimal solution $\tilde{c}_t^{[m]}$ for the terminal regions based on the loss function has no closed-form solution. Friedman et al. (2000) suggest an approximate solution by first order Taylor approximation.

In component-wise boosting the extension to distributional regression (Schmid et al., 2010; Mayr et al., 2012) allows handling multinomial loss as well as further complex loss functions.

*The R package `parboost` trains multiple boosting models independently on a subset of the data. This simple form of parallelism can be done with any machine learning algorithm and is not specific to boosting.

2.1.5. Component-wise Boosting for Distributional Regression

Extending component-wise boosting to distributional regression is a possible extension, since generalized additive models for location, scale and shape (GAMLSS) (Rigby and Stasinopoulos, 2005) are an extension of generalized additive models. The idea was originally proposed by Schmid et al. (2010) to handle overdispersion in count models, and later extended to general distributions by Mayr et al. (2012). So far, it has been assumed that the loss function depends on a single parameter that needs to be modeled. For negative log-likelihoods of distributions used as loss functions, this implies that only the location parameter, e.g., mean μ for a normal distribution, is modeled by the features. GAMLSS models extend this by making it possible to model multiple distribution parameters, e.g., the variance parameter σ^2 of a normal distribution. This is possible for an arbitrary number of parameters K. Current GAMLSS implements loss functions with up to six parameters (Stasinopoulos and Rigby, 2018). To extend the component-wise boosting framework to GAMLSS, an additional loop is introduced in the algorithm, which is shown in Algorithm 3. In general, separate hypothesis spaces $H_1, \ldots, H_K$ as well as maximum iteration numbers $m_{\text{stop},1}, \ldots m_{\text{stop},K}$ and learning rates can be defined for each $f_k(x)$. The original definition by Mayr et al. (2012) uses a fixed cycle to update the models $f_1(x), \ldots, f_K(x)$ of the algorithm. An alternative approach is discussed in Chapter 7 of this thesis. The software package `gamboostLSS` (Hofner et al., 2018) implements this method in the programming language R. It is build on top of `mboost` (Hothorn et al., 2018).

Algorithm 3: Gradient Boosting for distributional regression (Mayr et al., 2012). f_{-k} is a shorthand for $f_1, \ldots, f_{k-1}, f_{k+1}, \ldots, f_K$

Input: Data $\mathcal{D}$, hypothesis spaces $H_1, \ldots, H_K$, number of iterations $m_{\text{stop},1}, \ldots m_{\text{stop},K}$, learning rate ν.

Initialize $f_1^{[0]}, \ldots, f_K^{[0]}$ `// Initialization defined by loss and` y.

for $m \in \{1, \ldots, \max(m_{stop,1}, \ldots m_{stop,K})\}$ **do**

  for $k \in \{1, \ldots, K\}$ **do**

    if $m \leq m_{stop,k}$ **then**

$$r_k^{[m](i)} = -\left[\frac{\partial L\left(y^{(i)}, f\left(x^{(i)}\right), f_{-k}(x^{(i)})\right)}{\partial f\left(x^{(i)}\right)}\right]_{f=f_k^{[m-1]}} \qquad \forall i \in \{1, \ldots, n\}$$

$$h_k^{[m]} = \text{arg min}_{h \in H_k} \sum_{i=1}^{n} (r_k^{[m](i)} - h_k(x^{(i)}))$$

$$f_k^{[m]} := f_k^{[m-1]} + \nu h_k^{[m]}$$

return $f_1^{[m]}, \ldots, f_K^{[m]}$

2.2. Hyperparameter Tuning for Gradient Boosting

For nearly all machine learning algorithms, their hyperparameters have a critical impact on their performance and for optimal results, careful and often extensive tuning of these parameters is

required. For the general framework of hyperparameter optimization, a configuration space Λ has to be provided. It consists of parameters with (finite) ranges that can be sampled from. A single configuration $\lambda \in \Lambda$ induces a learner that can be trained on data $\mathcal{D}$, which is then usually evaluated using cross-validation. The aim of hyperparameter optimization is to find a $\lambda^* \in \Lambda$ that optimizes a performance measure.

Some algorithms like the random forest are relatively unaffected by their hyperparameters (Probst et al., 2018), but this is rather an exception. Figure A.1 in the appendix shows the performance distribution for different hyperparameter settings over 38 data sets for gradient boosting and random forest. It can be seen that for some of the data sets the distribution for gradient boosting is wider than the random forest, although there are still relevant improvements in the random forest by careful adjustment of its parameters. This is consistent with the results of Probst et al. (2018).

Especially modern efficient implementations of gradient tree boosting like *xgboost* or *lightGBM*, have a large number of hyperparameters. This makes their optimal tuning a challenge as it results in a high dimensional configuration space that has to be searched. For component-wise boosting, the number of hyperparameters normally tuned is usually much smaller and according to Bühlmann et al. (2007), only the number of iterations has to be tuned, since the learning rate is of relatively little importance as long as it is small enough. However, this is a theoretical statement and there are no published benchmarks to confirm this in practice. Additionally, if the parameters implemented by more complex base-learners are tuned and not set to their defaults or selected by human experts, a large number of additional hyperparameters is introduced. For example, a smooth P-spline base-learner would introduce up to seven additional hyperparameters (*degree, differences, knots, λ, degrees of freedom, cyclicality, monotonicity*). If each base-learner is configured separately for each feature, the dimension of the configuration space scales linearly with the number of features p, which will quickly become infeasible, especially for high-dimensional problems.

2.2.1. Bayesian Optimization

One of the state-of-the-art methods for hyperparameter optimization is *Bayesian optimization* (Jones et al., 1998; Snoek et al., 2012), also called *model-based optimization*. The main idea is to train a surrogate (regression) model on evaluated hyperparameter configurations, where cross-validated performance is the response. Since an evaluation is only a prediction of the surrogate model instead of a full cross-validation the response surface of this surrogate model can be optimized cost-effectively. The general sequence of Bayesian optimization is shown in Figure 2.1. An *infill-criterion* is optimized to balance the exploitation of estimated well-functioning configurations and the exploration of high uncertainty regions. These infill-criteria are usually composed of predicted performance and posterior model uncertainty. Therefore, models that allow uncertainty estimations such as Gaussian processes and random forests, are mainly used as surrogate models (Jones et al., 1998; Hutter et al., 2011). Their choice is often very problem-dependent, for example, in a fully numeric configuration space, Gaussian process surrogates are often outperforming random forests. In more complex spaces, with categorical and hierarchical hyperparameters,

random forests oftentimes outperform Gaussian processes. A benchmark of different Bayesian optimization implementations with their choices of surrogate models on the *hyperparameter optimization library* (Eggensperger et al., 2015) can be found in Chapter 5.

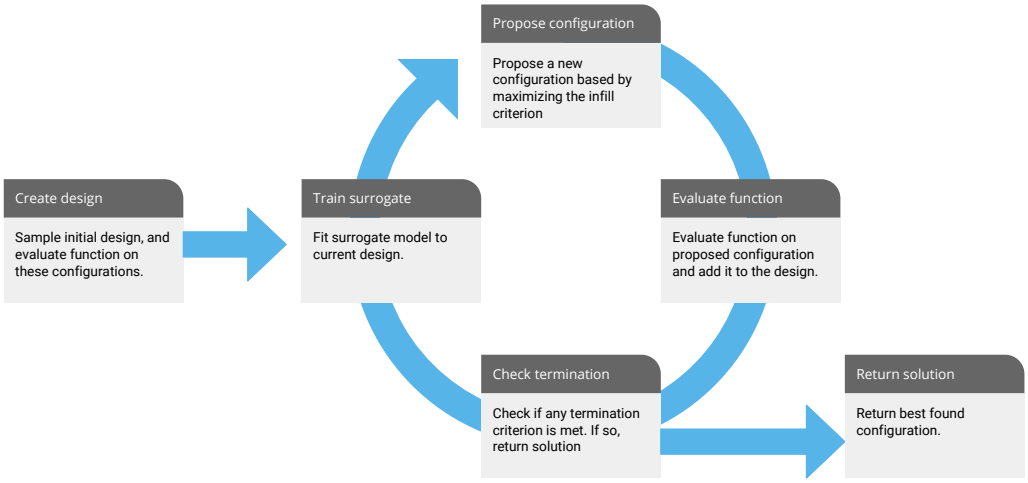

Figure 2.1.: Diagram of Bayesian optimization. A detailed description of every step in the process can be found in Chapter 5.

Bayesian optimization can be extended in multiple ways. Multi-point proposals, often referred to as batch Bayesian optimization, allow to propose multiple configurations in one iteration of the algorithm, which can save a large amount of time when multiple configurations can be evaluated in parallel. Important is the distinction between synchronous and asynchronous multi-point methods. The performance of the former can be severely reduced by heterogeneous runtime of the machine learning model to tune. In Chapter 6 an overview and benchmark of different multi-point strategies is given.

Often not only a single performance measure, but multiple measures have to be considered and optimized simultaneously. Useful applications in machine learning are for example the tuning of predictive performance and training time (Koch et al., 2012), false positive and negative rates (Horn and Bischl, 2016) or tuning of predictive performance and model sparseness. Several extensions of Bayesian optimization to the multi-objective case have been proposed (Jeong and Obayashi, 2005; Knowles, 2006; Bischl et al., 2014; Keane, 2006). A taxonomy of these methods can be found in Horn et al. (2015) and a benchmark of these methods is summarized in Chapter 5.

One of the usual assumptions of Bayesian optimization is that configurations can be evaluated noiselessly, i.e., without random error. This is clearly not a correct assumption in case of hyperparameter optimization, as the estimated performance based on cross-validation is always only

an estimation of the true generalization performance of the model. Several ways of correctly handling noisy evaluations in Bayesian optimization have been proposed. In general, there are methods that adapt surrogate modeling and infill criteria, e.g., Huang et al. (2006); Picheny et al. (2013), as well as methods that try to allocate budget optimally for reevaluation of configurations, e.g., Bartz-Beielstein et al. (2011, 2005). Building on this work gives the opportunity to dynamically choose the number of cross-validation folds for each configuration, which can save a large amount of computation resources.

2.2.2. Early Stopping

An important aspect of tuning any kind of gradient boosting model is the number of iterations. It is a special hyperparameter as it is possible to measure the performance for the full trace $1, \ldots, m_{\text{stop}}$ in a single fit. Algorithm 4 shows the general procedure to choose m optimally by *early stopping*. This approach has the drawback that we need validation data $\mathcal{D}_{\text{val}}$ to track the performance measured on out-of-bag data. Since in real applications data is always limited, the amount of training data $\mathcal{D}_{\text{train}}$ has to be reduced. More advanced approaches use cross-validation, which allows for more precise stopping, but suffer from additional computational costs.

Algorithm 4: Gradient Boosting with early Stopping

Input: Training data $\mathcal{D}_{\text{train}}$, early stopping data $\mathcal{D}_{\text{val}}$, performance measure $perf$, patience pat, hypothesis space H, maximum number of iterations m_{stop}, learning rate ν.

$f^{[0]} = \arg\min_c \sum_{i=1}^{n} L(y^{(i)}, c)$
$p^{[0]} = v(f^{[0]}(\mathbf{y}_{\text{val}}), \mathbf{y}_{\text{val}})$
$m := 0$
while $m \leq m_{stop}$ **do**
  $m := m + 1$
  $r^{[m](i)} = -\left[\frac{\partial L\left(y^{(i)}, f\left(x^{(i)}\right)\right)}{\partial f\left(x^{(i)}\right)}\right]_{f=f^{[m-1]}} \quad \forall i \in \{1, \ldots, n\}$
  $h^{[m]} = \arg\min_{h \in H} \sum_{i=1}^{n} (r^{[m](i)} - h(x^{(i)}))$
  $\beta^{[m]} = \arg\min_{\beta} \sum_{i=1}^{n} L(y^{(i)}, f^{[m-1]}(x^{(i)}) + \beta h^{[m]}(x^{(i)}))$
  $f^{[m]} = f^{[m-1]} + \nu \beta^{[m]} h^{[m]}$
  $p^{[m]} = perf(f^{[0]}(\mathbf{y}_{\text{val}}), \mathbf{y}_{\text{val}})$
  $m_{\text{opt}} = \arg\min_{m_c \in 1, \ldots, m} p^{[m_c]}$ ▷ w.l.o.g lower p is better.
  if $m_{opt} < m - pat$ **then**
    $m_{\text{stop}} := m_{\text{opt}}$
return $f^{[m_{opt}]}$

Early stopping focuses on predictive performance of the model to decide the optimal number of iterations. In component-wise boosting, sparseness is often a very desired property and early stopping is used here to reduce the number of features the model uses. If the sparseness and

identification of informative features is even more important, a *probing* approach can be used for early stopping. This is discussed in more detail in Subsection 2.4.2 and Chapter 5.

2.2.3. Hyperband

In general, any hyperparameter optimization strategy can be used for gradient boosting. But most advanced hyperparameter optimization strategies, like Bayesian optimization, are not able to handle the number of boosting iterations properly without adjustment. This means that the optimizer only has access to the final performance of the model, not the performance at each individual boosting iteration. Consequently, it is highly inefficient since a lot of information is discarded. A simple way of using this information is to stop training if the intermediate results are not promising compared to the results of other configuration at this training iteration. A structured way of doing this is the *successive halving* procedure of Jamieson and Talwalkar (2016). The main idea is to train a number of models for a small budget, i.e., iterations in gradient boosting, and drop the worst half. This happens iteratively with increasing budgets for each model until only a single model remains. Compared to training all models for their full budget, this approach can save a lot of computational resources. A major problem in this algorithm is the so called n *vs.* B/n question. Assuming a fixed budget B, the choice of the number of configurations to sample n has a very high impact on the result. For a large n, each configuration gets a smaller overall budget and many are stopped very early in training, which punishes configurations that have bad performance early on, but will become very strong if trained for a long enough time. For a very small n only a small fraction of the configuration space can be sampled, each getting a relatively high budget before potentially being dropped. The *Hyperband* algorithm (Li et al., 2017) addresses this problem by performing a grid search over n. The procedure is outlined as Algorithm 5.

Algorithm 5: General Hyperband Algorithm Li et al. (2017)

Input: max. resources R, discard fraction η

$s_{\max} = \lfloor \log_\eta(R) \rfloor$
$B = (s_{\max} + 1) \cdot R$
for $s \in \{s_{max}, s_{max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \cdot \frac{\eta^s}{(s+1)} \rceil$
  $r = R \cdot \eta^{-s}$
  $T = \text{sample\_configurations}(n)$
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n \cdot \eta^{-i} \rfloor$
    $r_i = r \cdot \eta^i$
    $k = \lfloor \frac{n_i}{\eta} \rfloor$
    $M = \{\text{train\_models\_and\_evaluate}(t, r_i) : t \in T\}$
    $T = \text{select\_top\_k}(T, L, k)$

Hyperband was initially mainly developed for the configuration of deep neural networks, but it is suitable for gradient boosting as well. It is also possible to apply it to any arbitrary machine learning algorithm configuration problem, if the budget is assumed to be the fraction of the full training data to use. Consequently, the models have to be retrained after each selection step, which is less effective but can still save a large amount of time as shown by Li et al. (2017). Hyperband has many desirable properties that make it a practical choice in many situations. It is easy to implement and understand, it can be parallelized efficiently and scaled up for massively parallel hyperparameter optimization, even asynchronously (Li et al., 2018). But a major drawback of Hyperband is, that the configuration space is only searched by a random search, which can be quite inefficient. Falkner et al. (2018) improved this by combining Hyperband with Bayesian optimization to search the configuration space much more efficient. A generic and expandable implementation of Hyperband in R can be found in the `hyperbandr`[†] package.

2.3. Machine Learning Pipelines

Up to this point the main focus was on the optimal configuration of the used machine learning algorithm. But for most real-world machine learning applications certain pre- and postprocessing operations are required or can greatly improve the model's performance. Preprocessing operations always depend on model type and data properties. This makes configuring a full machine learning pipeline a challenging problem. There are some simple preprocessing operations, like checking for (nearly) constant features, that should be done in any case. Further typical preprocessing operations that are considered by AutoML systems are feature scaling and transformation, feature filtering, imputation of missing values, dimensionality reduction, for example by principle component analysis, and encoding strategies for categorical features. Especially categorical features with a very high number of categories can be challenging since one-hot encoding is not a feasible approach anymore. Alternatives include clustering or merging of categories, impact encoding (Micci-Barreca, 2001), feature hashing (Moody, 1989) and numeric embeddings (Guo and Berkhahn, 2016). Many of these steps also add additional hyperparameters to the search space. Additionally, it is not given that a single encoding strategy is necessarily best for all categorical features in a data set. A useful trick is to define a maximum number of categories for features allowed to be one-hot encoded and to use a more sophisticated technique for features with more categories. This threshold has to be tuned by the AutoML system. Here the advantage of fully tree-based approaches can be seen as many of the above mentioned preprocessing steps become unnecessary. As discussed in Section 2.1.1, trees do not require any scaling or monotonous transformations as only the order of observations in features matters to the split finding algorithm. They also have methods to handle categorical features, missing values and possess feature selection properties. Solely in case of multi-class classification where no efficient optimal way for categorical features is known, different encoding methods have to be searched. In Chapter 9 the proposed AutoML framework tunes over a threshold for one-hot and impact encoding to handle categorical features. Typical postprocessing operations include tuning of classification thresholds (Cheng et al., 2011) and probability calibration (Niculescu-Mizil and Caruana, 2005).

[†] `https://github.com/ja-thomas/hyperbandr`

2.4. Feature Selection

In a variety of real-world applications, predictive performance of a model is not the only the deciding factor in model selection. Often, sparseness and interpretability are is crucial. Feature selection is a domain studied in statistics, machine learning, artificial intelligence and data mining (Guyon and Elisseeff, 2006). For certain applications, such as the analysis of gene expression data, it is a key component for analysis (Romero et al., 2006; Clarke et al., 2008).

While the main motivation for feature selection is often indeed the search for a sparser feature set, it can also help to improve predictive performance (Guyon and Elisseeff, 2006). In automatic machine learning, the focus is primarily on predictive performance, and in most approaches, sparseness is not rewarded. In other words, for two candidate models with (nearly) identical predictive performance the sparser model is not necessarily selected.

2.4.1. Feature Selection by Importance

The combination of models with feature selection techniques like filter (Bell and Wang, 2000) or wrapper methods (Aha and Bankert, 1996) often results in a higher computational effort. This makes machine learning algorithms that use automatic, inherent feature selection properties advantageous when sparseness is desired.

Gradient tree boosting consists of single regression trees that possess an automatic feature selection mechanism. This property of trees still applies for gradient tree boosting but in a less strict form. While each tree selects only some features to partition the space until a stopping criterion is reached, e.g., depth of the tree or minimum number of observations in a node, the added randomness from column and row subsampling yields a higher number of used features. Consequently, gradient tree boosting will often select many more features than a single tree.

One way to achieve sparser models with gradient tree boosting is to consider the relative importance of every feature. This works in the same way as other tree based methods (Breiman, 1984). It is defined as the overall empirical improvement in error or purity for all splits using a given feature. Finally, this is aggregated over all trees. Alternatives are the number of times a feature is used for splitting (Ke et al., 2017), the average number of samples that are affected by splits of a feature (Chen and Guestrin, 2016) or the decrease in performance based on random permutation of features (Breiman, 2001). While these approaches of feature importance only give a simple form of feature selection by giving all features that have never been used an importance of zero, it furthermore creates a ranking of all features by their importance, allowing the selection of the k most important features. These approaches have been criticized by Strobl et al. (2007) for random forests, and it is likely that similar arguments are valid for gradient tree boosting as well. Altmann et al. (2010) as well as Kursa et al. (2010) extend feature selection procedures based on feature importance with additional resampling for improved selection. Both of these methods require significant additional computational resources.

For component-wise boosting feature importance is only rarely used and less explored. This is because it has stronger feature selection properties, that are discussed in the following subsections. If feature importance is desired in component-wise boosting, it is straightforward to calculate by adding the loss reduction for groups of base-learners, e.g., all selected base-learners that use x_j. Calculating importance values for actual features instead of such groups of base-learners is much more difficult in cases where single base-learners contain multiple features.

2.4.2. Early Stopping for Sparseness

Usually, early stopping is triggered as soon as the empirical risk does not improve for a certain number of boosting iterations. Frequently this will result in the inclusion of many uninformative features (Meinshausen and Bühlmann, 2010; Mayr et al., 2012) for component-wise boosting. An easy way to solve this problem is the use of shadow features. These are features that have the same marginal distribution as a real feature, but are statistically independent from the target. Instead of early-stopping based on the out-of-bag risk, training can be stopped as soon as such a shadow feature is selected. This is further discussed in Chapter 4.

The recent introduction of *knockoffs* (Barber et al., 2015; Candes et al., 2018) enables the creation of better shadow features that not only have the same marginal distribution, but also identical joint distribution with all other features in the data set. Knockoffs have already successfully been applied to control the false discovery rate in stepwise feature selection. The biggest disadvantage of using knockoffs is that they are difficult to create and usually require a known data distribution $\mathcal{X}$ or complex generative procedures (Jordon et al., 2019).

2.4.3. Stability of Feature Selection

Controlling the number of selected uninformative features, that is, features with no dependency to the target is an important aspect in feature selection, as it gives a principle to choose levels of regularization and consequently number of features to select. Meinshausen and Bühlmann (2010) introduced *stability selection*, a method to control the false discovery rate for a number of feature selection methods. Their approach was later refined by Shah and Samworth (2013) and shown to work well for component-wise boosting by Hofner et al. (2015). The general idea is to repeatedly subsample the data and apply the same feature selection method to each subsample. Only features with selection frequencies of at least π_{thr} are considered in the end. Combined with a feature limit for the selection methods q, an upper bound of the *per-family error rate* $\mathbb{E}(V)$ can be calculated, where V is the number of selected uninformative features.

$$\mathbb{E}(V) = \frac{q^2}{(2\pi_{\text{thr}} - 1)p} \tag{2.10}$$

Here, p is the number of features in $\mathcal{D}$. Algorithm 6 shows the general procedure of applying stability selection to gradient boosting.

Algorithm 6: Stability Selection for Gradient Boosting (Hofner et al., 2015)

Input: Data $\mathcal{D}$, hypothesis space H, number of iterations m_{stop}, learning rate ν, maximum number of selected features per model q, selection threshold π_{thr}.

for $b \in \{1, \ldots, B\}$ **do**
  Subsample $\mathcal{D}_b$ of size $\lfloor n/2 \rfloor$ from $\mathcal{D}$.
  Initialize $f_b^{[0]} = \arg\min_c \sum_{y^{(i)} \in \mathcal{D}_b} L(y^{(i)}, c)$.
  for $m \in \{1, \ldots, m_{stop}\}$ **do**
    update pseudo residuals and choose new base learner on $\mathcal{D}_b$
    $f_b^{[m]} := f_b^{[m-1]} + \nu \beta^{[m]} h^{[m]}$
    $\hat{S}_b = \text{get\_selected\_features}(f_b^{[m]})$
    if $|\hat{S}_b| \geq q$ **then** `// count number of features in` $f_b^{[m]}$
      break
for $j \in \{1, \ldots, p\}$ **do**
  $\hat{\pi}_j = \frac{1}{B} \sum_{b=1}^{B} \mathbb{I}_{\{j \in \hat{S}_b\}}$
$\hat{S}_{\text{stable}} = \{j : \hat{\pi}_j \geq \pi_{\text{thr}}\}$
return $\hat{S}_{stable}$

A major drawback of this approach is that of $q, \mathbb{E}(V)$ and π_{thr}, two have to be chosen by the user (the missing value can be calculated from Equation 2.10). In Chapter 4 a comparison between stability selection, early stopping using shadow features and regular cross-validation is discussed.

3. Future Direction and Open Questions

Automatic machine learning is a growing area that helps to meet the rapidly growing demand for machine learning experts. In this chapter three open problem and research directions are briefly discussed. These improvements should enable AutoML to be applicable in more real-world applications.

AutoML becomes more powerful and is made easier to use by companies providing AutoML services, such as Google's *Cloud AutoML* and Amazon's *Sagemaker*. These simplify the deployment of trained AutoML systems in a business environment and allow their scaling to millions of users. It is important to note that the lifecycle of a machine learning model does not end with deployment. Changes in the data distribution or model requirements can be devastating, if they are not detected and handled appropriately. Worse still, the deployment and use of a machine learning model itself can change the future data distribution as different business decisions are made based on the models predictions. This results in a need for repeated model selection and adaption. If the goal of automatic machine learning is to automate the complete machine learning model lifecycle, detecting drifts and monitoring aspects need to be automated as well. A good starting point was the third AutoML challenge by *chalearn* called *Lifelong AutoML*[*]. It included changes in data distribution by concept-drift the AutoML systems had to adapt to. First approaches for this like Wilson et al. (2018) and Madrid et al. (2018) have been introduced, and hopefully more work will follow.

A further important aspect is that current research places a great deal of emphasis on model performance, i.e., for most benchmarks and challenges a single measure has to be optimized. In practice, there are often many obvious or hidden constraints in the model selection process. Current AutoML approaches are not flexible enough to reflect and support this. Examples for obvious model constraints include model size, which can be important when models should be deployed on end-user devices, or prediction speed, when decisions need to be made in fractions of a second. These constraints could be handled by multi-objective optimization which build a Pareto front of solutions and lets the domain expert decide which trade-off between different measures is best for a given problem. A more challenging problem arises when model requirements are more difficult to quantify. Interpretability is an important aspect for many real world applications of machine learning. Many see explainable machine learning as one of the most important and promising fields in machine learning and a large amount of funding is put in this area like the

[*] https://competitions.codalab.org/competitions/19836

Defense Advanced Research Projects Agency (DARPA) with their Explainable Artificial Intelligence (XAI) program started in 2016. The optimization of AutoML for interpretability is still an open problem. The simple solution to solely tune over interpretable models is not good enough, since compromises between model performance and interpretability are inherent, and ultimately, these still have to be made by humans with domain knowledge. The problem becomes even more difficult when constrains are not formally specified or known beforehand. The inclusion of humans back in the model selection process seems like a step back for AutoML at first, but extensions to a human-in-the-loop AutoML system would allow model selection in settings with unknown constrains and preferences. The human in this approach does not need to be an expert in machine learning anymore, but only be able to decide which aspects of machine learning models, presented to him by an AutoML system in an easy to understand way, are to be preferred.

If machine learning competitions such as *kaggle* are a good indication of the state-of-the-art in supervised machine learning, it becomes apparent that one of the most important aspects is still feature engineering. Feature engineering, i.e., the combination of several existing features to create more informative ones and improve predictive performance, is still a mostly manual process. Approaches to automatic feature engineering exist, e.g. Kanter and Veeramachaneni (2015) and Nargesian et al. (2017), but the problem still suffers from exponential growth in possible combinations of features in higher dimensional data. A human-in-the-loop AutoML approach could be proposed here as well. Domain experts usually understand their data very well and can define features that belong together semantically, but do not know in which way to combine these feature most efficiently. In such a reduced search space automatic feature engineering could be done quite efficiently.

The main task of AutoML should be the automation of workflows that can be efficiently optimized by a machine, while still allowing repeated user feedback with information that is hard, or even impossible, for the machine to learn autonomously.

The rest of this thesis is structured as following. Chapters 4 to 10 contain peer-reviewed publications, technical reports and workshop contributions (co-)authored by the author of this thesis in chronological order of their publication. The individual contributions of each author of these publications is described separately in the beginning of the respective chapters.

4. Probing for Sparse and and Fast Variable Selection with Model-Based Boosting

Declaration of the specific contributions of the author The publication *Probing for Sparse and Fast Variable Selection with Model-Based Boosting* was done in equal parts by the author and Tobias Hepp. Disentangling the specific contributions is difficult as both authors did multiple iterations over the text and conducted the necessary benchmarks. The implementation of the proposed algorithm in the R-package `mboost` was done by the author, as well as the initial benchmarking code. The idea for the paper was proposed by Bernd Bischl, who guided the research process. Andreas Mayr helped improving the general methods section and all other parts of the paper substantially.

Hindawi
Computational and Mathematical Methods in Medicine
Volume 2017, Article ID 1421409, 8 pages
https://doi.org/10.1155/2017/1421409

Research Article

Probing for Sparse and Fast Variable Selection with Model-Based Boosting

Janek Thomas,[1] Tobias Hepp,[2] Andreas Mayr,[2,3] and Bernd Bischl[1]

[1]*Department of Statistics, LMU München, München, Germany*
[2]*Department of Medical Informatics, Biometry and Epidemiology, FAU Erlangen-Nürnberg, Erlangen, Germany*
[3]*Department of Medical Biometry, Informatics and Epidemiology, University Hospital Bonn, Bonn, Germany*

Correspondence should be addressed to Tobias Hepp; tobias.hepp@uk-erlangen.de

Janek Thomas and Tobias Hepp contributed equally to this work.

Received 9 February 2017; Accepted 13 April 2017; Published 31 July 2017

Academic Editor: Yuhai Zhao

We present a new variable selection method based on model-based gradient boosting and randomly permuted variables. Model-based boosting is a tool to fit a statistical model while performing variable selection at the same time. A drawback of the fitting lies in the need of multiple model fits on slightly altered data (e.g., cross-validation or bootstrap) to find the optimal number of boosting iterations and prevent overfitting. In our proposed approach, we augment the data set with randomly permuted versions of the true variables, so-called shadow variables, and stop the stepwise fitting as soon as such a variable would be added to the model. This allows variable selection in a single fit of the model without requiring further parameter tuning. We show that our probing approach can compete with state-of-the-art selection methods like stability selection in a high-dimensional classification benchmark and apply it on three gene expression data sets.

1. Introduction

At the latest since the emergence of genomic and proteomic data, where the number of available variables p is possibly far higher than the sample size n, high-dimensional data analysis becomes increasingly important in biomedical research [1–4]. Since common statistical regression methods like ordinary least squares are unable to estimate model coefficients in these settings due to singularity of the covariance matrix, varying strategies have been proposed to select only truly influential, that is, informative, variables and discard those without impact on the outcome.

By enforcing sparsity in the true coefficient vector, regularized regression approaches like the *lasso* [5], *least angle regression* [6], *elastic net* [7], and *gradient boosting* algorithms [8, 9] perform variable selection directly in the model fitting process. This selection is controlled by tuning hyperparameters that define the degree of penalization. While these hyperparameters are commonly determined using resampling strategies like cross-validation, bootstrapping, and similar methods, the focus on minimizing the prediction error often results in the selection of many noninformative variables [10, 11].

One approach to address this problem is *stability selection* [12, 13], a method that combines variable selection with repeated subsampling of the data to evaluate selection frequencies of variables. While stability selection can considerably improve the performance of several variable selection methods including regularized regression models in high-dimensional settings [12, 14], its application depends on additional hyperparameters. Although recommendations for reasonable values exist [12, 14], proper specification of these parameters is not straightforward in practice as the optimal configuration would require a priori knowledge about the number of informative variables. Another potential drawback is that stability selection increases the computational demand, which can be problematic in high-dimensional settings if the computational complexity of the used selection technique scales superlinearly with the number of predictor variables.

In this paper, we propose a new method to determine the optimal number of iterations in model-based boosting for variable selection inspired by *probing*, a method frequently used in related areas of machine learning research [15–17] and the analysis of microarrays [18]. The general notion of probing involves the artificial inflation of the data with random noise variables, so-called *probes* or *shadow variables*. While this approach is in principle applicable to the lasso or least angle regression as well, it is especially attractive to use with more computationally intensive boosting algorithms, as no resampling is required at all. Using the first selection of a shadow variable as stopping criterion, the algorithm is applied only once without the need to optimize any hyperparameters in order to extract a set of informative variables from the data, thereby making its application very fast and simple in practice. Furthermore, simulation studies show that the resulting models in fact tend to be more strictly regularized compared to the ones resulting from cross-validation and contain less uninformative variables.

In Section 2, we provide detailed descriptions of the model-based gradient boosting algorithm as well as stability selection and the new probing approach. Results of a simulation study comparing the performance of probing to cross-validation and different configurations of stability selection in a binary classification setting are then presented in Section 3 before discussing the application of these methods on three data sets with measurements of gene expression levels in Section 4. Section 5 summarizes our findings and presents an outlook to extensions of the algorithm.

2. Methods

2.1. Gradient Boosting. Given a learning problem with a data set $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1,\ldots,n}$ sampled i.i.d. from a distribution over the joint space $\mathcal{X} \times \mathcal{Y}$, with a p-dimensional input space $\mathcal{X} = (\mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_p)$ and an output space $\mathcal{Y}$ (e.g., $\mathcal{Y} = \mathbb{R}$ for regression and $\mathcal{Y} = \{0, 1\}$ for binary classification), the aim is to estimate a function, $f(\mathbf{x})$, $\mathcal{X} \to \mathcal{Y}$, that maps elements of the input space to the output space as good as possible. Relying on the perspective on boosting as gradient descent in function space, gradient boosting algorithms try to minimize a given loss function, $\rho(y^{(i)}, f(\mathbf{x}^{(i)}))$, $\rho : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$, that measures the discrepancy between a predicted outcome value of $f(\mathbf{x}^{(i)})$ and the true $y^{(i)}$. Minimizing this discrepancy is achieved by repeatedly fitting weak prediction functions, called *base learners*, to previous mistakes, in order to combine them to a strong ensemble [19]. Although early implementations in the context of machine learning focused specifically on the use of regression trees, the concept has been successfully extended to suit the framework of a variety of statistical modelling problems [8, 20]. In this model-based approach, the *base learners* $h(\mathbf{x})$ are typically defined by semiparametric regression functions on $\mathbf{x}$ to build an additive model. A common simplification is to assume that each base learner h_j is defined on only one component x_j of the input space

$$f(\mathbf{x}) = \beta_0 + h_1(x_1) + \cdots + h_p(x_p). \tag{1}$$

For an overview of the fitting process of model-based boosting see Algorithm 1.

Algorithm 1 (model-based gradient boosting). Starting at $m = 0$ with a constant loss minimal initial value $\widehat{f}^{[0]}(\mathbf{x}) \equiv c$, the algorithm iteratively updates the predictor with a small fraction of the base learner with the best fit on the negative gradient of the loss function:

(1) Set iteration counter $m := m + 1$.

(2) While $m \leq m_{\text{stop}}$, compute the negative gradient vector of the loss function:

$$u^{(i)} = -\left.\frac{\partial \rho(y, f)}{\partial f}\right|_{f=\widehat{f}^{[m-1]}(\mathbf{x}^{(i)}), y=y^{(i)}}. \tag{2}$$

(3) Fit every base learner $h_j^{[m]}(x_j)$ separately to the negative gradient vector $\mathbf{u}$.

(4) Find $\widehat{h}_{j^*}^{[m]}(\mathbf{x}_{j^*})$, that is, the base learner with the best fit:

$$j^* = \arg\min_{1 \leq j \leq p} \sum_{i=1}^{n} \left(u^{(i)} - \widehat{h}_j^{[m]}\left(x_j^{(i)}\right)\right)^2. \tag{3}$$

(5) Update the predictor with a small fraction $0 \leq \nu \leq 1$ of this component:

$$\widehat{f}(\mathbf{x})^{[m]} = \widehat{f}(\mathbf{x})^{[m-1]} + \nu \cdot \widehat{h}_{j^*}^{[m]}\left(x_{j^*}\right). \tag{4}$$

The resulting model can be interpreted as a generalized additive model with partial effects for each covariate contained in the additive predictor. Although the algorithm relies on two hyperparameters ν and m_{stop}, Bühlmann and Hothorn [9] claim that the *learning rate* ν is of minor importance as long as it is "sufficiently small," with $\nu = 0.1$ commonly used in practice.

The stopping criterion, m_{stop}, determines the degree of regularization and thereby heavily affects the model quality in terms of overfitting and variable selection [21]. However, as already outlined in the introduction, optimizing m_{stop} using common approaches like cross-validation results in the selection of many uninformative variables. Although still focusing on minimizing prediction error, using a 25-fold bootstrap instead of the commonly used 10-fold cross-validation tends to return sparser models without sacrificing prediction performance [22].

2.2. Stability Selection. The weak performance of cross-validation regarding variable selection partly results from the fact that it pursues the goal of minimizing the prediction error instead of selecting only informative variables. One possible solution is the *stability selection* framework [12, 13], a very versatile algorithm that can be combined with all kinds of variable selection methods like gradient boosting, lasso, or forward stepwise selection. It produces sparser solutions by controlling the number of false discoveries. Stability selection defines an upper bound for the per-family error rate (PFER),

for example, the expected number of uninformative variables $\mathbb{E}(V)$ included in the final model.

Therefore, using stability selection with model-based boosting means that Algorithm 1 is run independently on B random subsamples of the data until either a predefined number of iterations m_{stop} is reached or q different variables have been selected. Subsequently, all variables are sorted with respect to their selection frequency in the B sets. The amount of informative variables is then determined by a user-defined threshold π_{thr} that has to be exceeded. A detailed description of these steps is given in Algorithm 2.

Algorithm 2 (stability selection for model-based boosting [14]).

(1) For $b = 1, \ldots, B$,

  (a) draw a subset of size $\lfloor n/2 \rfloor$ from the data;
  (b) fit a boosting model to the subset until the number of selected variables is equal to q or the number of iterations reaches a prespecified number (m_{stop}).

(2) Compute the selection frequencies per variable j:

$$\widehat{\pi}_j := \frac{1}{B} \sum_{b=1}^{B} \mathbb{I}_{\{j \in \widehat{S}_b\}}, \tag{5}$$

where $\widehat{S}_b$ denotes the set of selected variables in iteration b.

(3) Select variables with a selection frequency of at least π_{thr}, which yields a set of stable covariates:

$$\widehat{S}_{\text{stable}} := \left\{ j : \widehat{\pi}_j \geq \pi_{\text{thr}} \right\}. \tag{6}$$

Following this approach, the upper bound for the PFER can be derived as follows [12]:

$$\mathbb{E}(V) \leq \frac{q^2}{(2\pi_{\text{thr}} - 1)\, p}. \tag{7}$$

With additional assumptions on exchangeability and shape restrictions on the distribution of simultaneous selection, even tighter bounds can be derived [13]. While this method is successfully applied in a large number of different applications [23–26], several shortcomings impede the usage in practice. First off, three additional hyperparameters π_{thr}, PFER, and q are introduced. Although only two of them have to be specified by the user (the third one can be calculated by assuming equality in (7)), it is not intuitively clear which parameter should be left out and how to specify the remaining two. Even though recommendations for reasonable settings for the selection threshold [12] or the PFER [14] are proposed, the effectiveness of these settings is difficult to evaluate in practical settings. The second obstacle in the usage of stability selection is the considerable computational power required for calculation. Overall B boosting models ([13] recommends $B = 100$) have to be fitted and a reasonable m_{stop} has to be found as well, which will most likely require cross-validation. Even though this process can be parallelized quite easily, complex model classes with smooth and higher-order effects can become extremely costly to fit.

2.3. Probing. The approach of adding *probes* or *shadow variables*, for example, artificial uninformative variables to the data, is not completely new and has already been investigated in some areas of machine learning. Although they share the underlying idea to benefit from the presence of variables that are known to be independent from the outcome, the actual implementation of the concept differs (see Guyon and Elisseeff (2003) [15] for an overview). An especially useful approach, however, is to generate these additional variables as randomly shuffled versions of all observed variables. These permuted variables will be called *shadow variables* for the remainder of this paper and are denoted as $\tilde{x}_j$. Compared to adding randomly sampled variables, shadow variables have the advantage that the marginal distribution of x_j is preserved in $\tilde{x}_j$. This approach is tightly connected to the theory of permutation tests [27] and is used similarly for *all-relevant* variable selection with random forests [28].

Implementing the *probing* concept to the sequential structure of model-based gradient boosting is rather straightforward. Since boosting algorithms proceed in a greedy fashion and only update the effect which yields the largest loss reduction in each iteration, selecting a shadow variable essentially implies that the best possible improvement at this stage relies on information that is known to be unrelated to the outcome. As a consequence, variables that are selected in later iterations are most likely correlated to y only by chance as well. Therefore, all variables that have been added prior to the first shadow variable are assumed to have a true influence on the target variable and should be considered informative. A description of the full procedure is presented in Algorithm 3.

Algorithm 3 (probing for variable selection in model-based boosting).

(1) *Expand* the data set X by creating randomly shuffled images $\tilde{x}_j$ for each of the $j = 1, \ldots, p$ variables x_j such that

$$\tilde{x}_j \in S_{x_j}, \tag{8}$$

where S_{x_j} denotes the symmetric group that contains all $n!$ possible permutations of x_j.

(2) *Initialize* a boosting model on the inflated data set

$$\overline{X} = \left[x_1 \cdots x_p \; \tilde{x}_1 \cdots \tilde{x}_p \right] \tag{9}$$

and start iterations with $m = 0$.

(3) *Stop if* the first $\tilde{x}_j$ is selected; see Algorithm 1 step (3).

(4) *Return* only the variables selected from the original data set X.

The major advantage of this approach compared to variable selection via cross-validation or stability selection is that one model fit is enough to find informative variables and no expensive refitting of the model is required. Additionally, there is no need for any prespecification like the search space (m_{stop}) for cross-validation or additional hyperparameters (q, π_{thr}, PFER) for stability selection. However, it should be noted that, unlike classical cross-validation, probing aims at optimal variable selection instead of prediction performance of the algorithm. Since this usually involves stopping much earlier, the effect estimates associated with the selected variables are most likely strongly regularized and might not be optimal for predictions.

3. Simulation Study

In order to evaluate the performance of our proposed variable selection method, we conduct a benchmark simulation study where we compare the set of nonzero coefficients determined by the use of shadow variables as stopping criterion to cross-validation and different configurations of stability selection. We simulate n data points for p variables from a multivariate normal distribution $X \sim \mathcal{N}(0, \Sigma)$ with Toeplitz correlation structure $\Sigma_{ij} = \rho^{|i-j|}$ for all $1 < i, j < p$ and $\rho = 0.9$. The response variable $y^{(i)}$ is then generated by sampling Bernoulli experiments with probability

$$\pi^{(i)} = \frac{\exp\left(\eta^{(i)}\right)}{1 + \exp\left(\eta^{(i)}\right)}, \tag{10}$$

with $\eta^{(i)}$ the linear predictor for the ith observation $\eta^{(i)} = X^{(i)}\beta$ and all nonzero elements of β sampled from $\mathcal{U}(-1, 1)$. Since the total amount of nonzero coefficients determines the number of informative variables in the setting, it is denoted as p_{inf}.

Overall, we consider 12 different simulation scenarios defined by all possible combinations of $n \in \{100, 500\}$, $p \in \{100, 500, 1000\}$, and $p_{\text{inf}} \in \{5, 20\}$. Specifically, this leads to the evaluation of 2 low-dimensional settings with $p < n$, 4 settings with $p = n$, and 6 high-dimensional settings with $p > n$. Each configuration is run 100 times. Along with new realizations of X and y, we also draw new values for the nonzero coefficients in β and sample their position in the vector in each run to allow for varying correlation patterns among the informative variables. For variable selection with cross-validation, 25-fold bootstrap (the default in `mboost`) is used to determine the final number of iterations. Different configurations of stability selection were tested to investigate whether and, if so, to what extent these settings affect the selection. In order to explicitly use the upper error bounds of stability selection, we decided to specify 9 combinations with PFER $\in \{1, 2.5, 8\}$ and $\pi_{\text{thr}} \in \{0.6, 0.75, 0.9\}$ and calculate q from (7). Aside from the learning rate ν, which is set to 0.1 for all methods, no further parameters have to be specified for the probing scheme. Two performance measures are considered for the evaluation of the methods with respect to variable selection: first, the true positive rate (TPR) as the fraction of (correctly) selected variables from all true informative variables and, second, the false discovery rate (FDR) as the fraction of uninformative variables in the set of selected variables. To ensure reproducibility the R package `batchtools` [29] was used for all simulations.

The results of the simulations for all settings are illustrated in Figure 1. With TPR and FDR on the y-axis and x-axis, respectively, solutions displayed in the top left corner of the plots therefore successfully separate p_{inf} informative variables from the ones without true effect on the response. Although already using a sparse cross-validation approach, the FDR of variable selection via cross-validation is still relatively high, with more than 50% false positives in the selected sets in the majority of the simulated scenarios. Whereas this seems to be mostly disadvantageous in the cases where $p_{\text{inf}} = 5$, the trend to more greedy solutions leads to a considerably higher chance of identifying more of the truly informative variables if $p_{\text{inf}} = 20$ or with very high p, however, still at the price of picking up many noise variables on the way. Pooling the results of all configurations considered for stability selection, the results cover a large area of the performance space in Figure 1, thereby probably indicating high sensitivity on the decisions regarding the three tuning parameters.

Examining the results separately in Figure 2, the dilemma is particularly clearly illustrated for $p_{\text{inf}} = 20$ and $n = 500$. Despite being able to control the upper bounds for expected false positive selections, only a minority of the true effects are selected if the PFER is set too conservative. In addition, the high variance of the FDR observed for these configurations in some settings somewhat counteracts the goal to achieve more certainty about the selected variables one might probably pursue by setting the PFER very low. The performance of probing, on the other hand, reveals a much more stable pattern and outperforms stability selection in the difficult $p_{\text{inf}} = 20$ and $n = 100$ settings. In fact, the TPR is either higher or similar to all configurations used for stability selection, but exhibiting slightly higher FDR especially in settings with $n = 500$. Interestingly, probing seems to provide results similar to those of stability selection with PFER = 8, raising the question if the use of shadow variables allows statements about the number of expected false positives in the selected variable set.

Considering the runtime, however, we can see that probing is orders of magnitudes faster with an average runtime of less than a second compared to 12 seconds for cross-validation and almost one minute for stability selection.

4. Application on Gene Expression Data

In this section we exploit the usage of probing as a tool for variable selection on three gene expression data sets. More specifically, this includes data from using oligonucleotide arrays for colon cancer detection [30] with 40 tumor and 22 regular colon tissue samples and $p = 2000$ measured genes expression levels. In addition, we analyse data from a study aiming to predict metastasis of breast carcinoma [31], where patients were labelled good or poor ($n = 111$ and $n = 57$, resp.) depending on whether they remained event-free for a five-year period after diagnosis or not. The data set contains log-transformed expression levels of $p =$

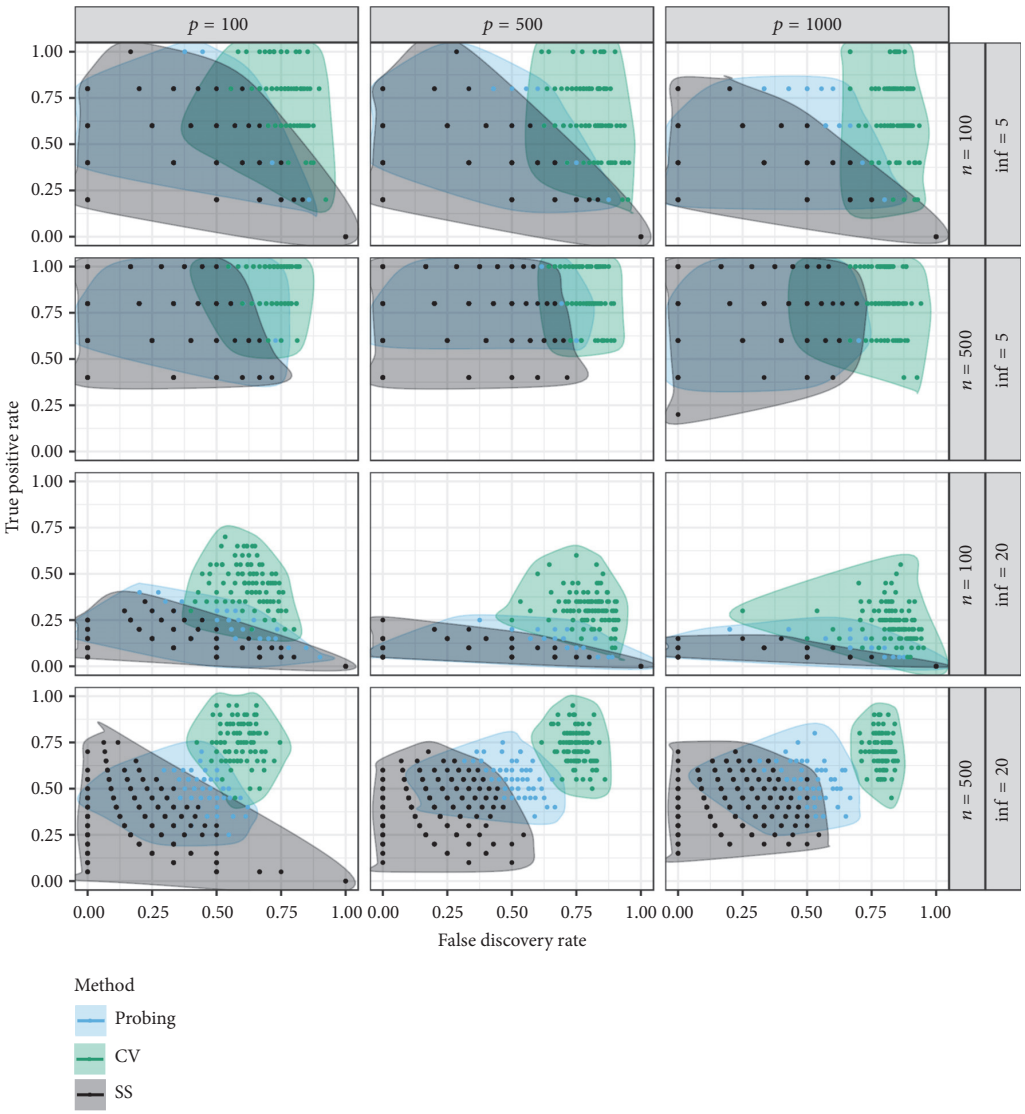

FIGURE 1: True positive rate (on y-axis) and false discovery rate (on x-axis) for three different, boosting-based variable selection algorithms, probing (black), stability selection (green), cross-validation (blue), and different simulation settings: $n \in \{100, 500\}$, $p \in \{100, 500, 1000\}$, and $p_{\text{inf}} \in \{5, 20\}$. All settings of stability selection are combined. Shaded areas are smooth hulls around all observed values.

2905 genes. The last example examines riboflavin production by *Bacillus subtilis* [32] with $n = 71$ observations of log-transformed riboflavin production rates and expression level for $p = 4088$ genes. All data are publicly available via `R` packages `datamicroarray` and `hdi`. Our proposed probing approach is implemented in a fork of the `mboost` [33] software for component-wise gradient boosting. It can be easily used by setting `probe=TRUE` in the `glmboost()` call.

In order to evaluate the results provided by the new approach, we analysed the data using cross-validation, stability selection [34], and the lasso [35] for comparison. Table 1 shows the total number of variables selected by each method along with the size of the intersection between the sets. Starting with the probably least surprising result, boosting with cross-validation leads to the largest set of selected variables in all examples, whereas using probing as stopping criterion instead clearly reduces these sets. Since both approaches are based on the same regularization profile until the first shadow variable enters the model, the less regularized solution of cross-validation always contains all variables selected with probing. For stability selection, we used the conservative approach with PFER = 1 and $q = 20$ as suggested by Bühlmann et al. (2014) [32]. As a consequence, the set of variables considered to be informative further

Table 1: Total number of selected variables and intersection size for four variable selection techniques (boosting with 25-fold bootstrap, probing, stability selection, and the lasso with 10-fold cross-validation) on three gene expression data sets. The last column compares algorithm runtime in seconds.

	Cross-validation	Probing	Stability selection	Lasso (`glmnet`)	Runtime (sec.)
			Colon cancer		
Cross-validation	**9**				10.52
Probing	5	**5**			1.78
Stability selection	3	3	**3**		49.4
Lasso (`glmnet`)	7	5	3	**7**	0.4
			Breast carcinoma		
Cross-validation	**32**				24
Probing	14	**14**			4.39
Stability selection	1	1	**1**		102.28
Lasso (`glmnet`)	14	14	1	**14**	1.13
			Riboflavin production		
Cross-validation	**50**				14.2
Probing	10	**10**			6.89
Stability selection	5	5	**5**		66.46
Lasso (`glmnet`)	23	7	4	**30**	0.68

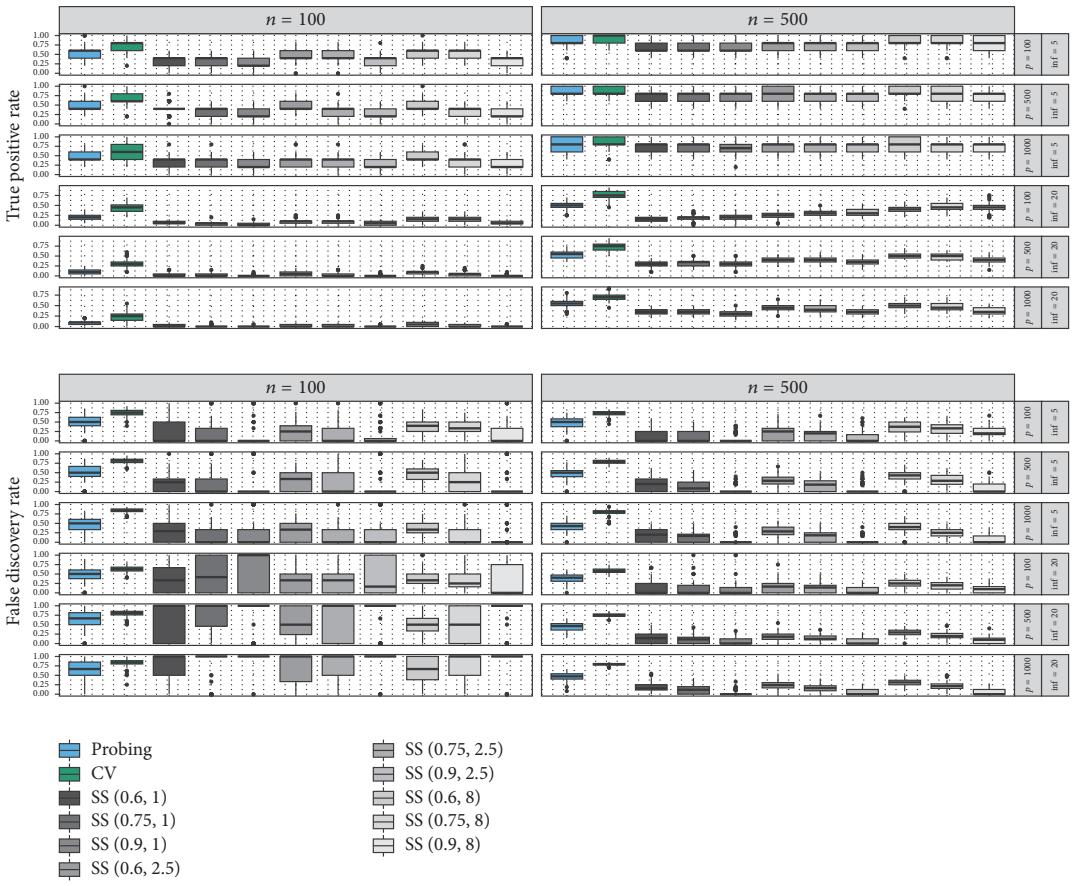

Figure 2: Boxplots of true positive rate (top) and false discovery rate (bottom) for different simulation settings and the three boosting-based, variable selection algorithms. Different Stability selection settings are denoted by SS(π_{thr}, PFER).

shrinks in all three scenarios. Again, these results clearly reflect the findings from the simulation study in Section 3, placing the probing approach between stability selection with probably overly conservative error bound and the greedy selection with cross-validation.

Since so far all approaches rely on boosting algorithms, we additionally considered variable selection with the lasso. We used the default settings of the `glmnet` package for R to calculate the lasso regularization path and determine the final model via 10-fold cross-validation [35]. Although the lasso already tends to result in sparser models under these conditions compared to model-based boosting [22], `glmnet` additionally uses a "one-standard-error rule" to regularize the solution even further. In fact, this leads to the selection of an identical set of genes as probing for the breast carcinoma example, but the final models estimated for both other examples still contain a higher number of variables. This is especially the case for the data on riboflavin production, where the lasso solution is further not simply a subset of the cross-validated boosting approach and only agrees on 23 mutually selected variables. Interestingly, even one of the 5 variables proposed by stability selection is also missing. The R code used for this analysis can be found in the Supplementary Material of this manuscript available online at https://doi.org/10.1155/2017/1421409.

5. Conclusion

We proposed a new approach to determine the optimal number of iterations for sparse and fast variable selection with model-based boosting via the addition of probes or shadow variables (*probing*). We were able to demonstrate via a simulation study and the analysis of gene expression data that our approach is both a feasible and convenient strategy for variable selection in high-dimensional settings. In contrast to common tuning procedures for model-based boosting which rely on resampling or cross-validation procedures to optimize the prediction accuracy [21], our probing approach directly addresses the variable selection properties of the algorithm. As a result, it substantially reduces the high number of false discoveries that arise with standard procedures [14] while only requiring a single model fit to obtain the set of parameters.

Aside from the very short runtime, another attractive feature of probing is that no additional tuning parameters have to be specified to run the algorithm. While this greatly increases its ease of use, there is, of course, a trade-off regarding flexibility, as the lack of tuning parameters means that there is no way to steer the results towards more or less conservative solutions. However, a corresponding tuning approach in the context of probing could be to allow a certain amount of selected probes in the model before deciding to stop the algorithm (cf. Guyon and Elisseeff, 2003 [15]). Although variables selected after the first probe can be labelled informative less convincingly, this resembles the uncertainty that comes with specifying higher values for the error bound of stability selection.

A potential drawback of our approach is that due to the stochasticity of the permutations, there is no deterministic solution and the selected set might slightly vary after rerunning the algorithm. In order to stabilize results, probing could also be used combined with resampling to determine the optimal stopping iteration for the algorithm by running the procedure on several bootstrap samples first. Of course, this requires the computation of multiple models and therefore again increases the runtime of the whole selection procedure.

Another promising extension could be a combination with stability selection. With each model stopping at the first shadow variable, only the selection threshold π_{thr} has to be specified. However, since this means a fundamental change of the original procedure, further research on this topic is necessary to better assess how this could affect the resulting error bound.

While in this work we focused on gradient boosting for binary and continuous data, there is no reason why our results should not also carry over to other regression settings or related statistical boosting algorithms as likelihood-based boosting [36]. Likelihood-based boosting follows the same principle idea but uses different updates, coinciding with gradient boosting in case of Gaussian responses [37]. Further research is also warranted on extending our approach to multidimensional boosting algorithms [25, 38], where variables have to be selected for various models simultaneously.

In addition, probing as a tuning scheme could be generally also combined with similar regularized regression approaches like the lasso [5, 22]. Our proposal for model-based boosting hence could be a starting point for a new way of tuning algorithmic models for high-dimensional data, not with the focus on prediction accuracy, but addressing directly the desired variable selection properties.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work of authors Tobias Hepp and Andreas Mayr was supported by the Interdisciplinary Center for Clinical Research (IZKF) of the Friedrich-Alexander-University Erlangen-Nürnberg (Project J49). The authors additionally acknowledge support by Deutsche Forschungsgemeinschaft and Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) within the funding programme Open Access Publishing.

References

[1] R. Romero, J. Espinoza, F. Gotsch et al., "The use of high-dimensional biology (genomics, transcriptomics, proteomics, and metabolomics) to understand the preterm parturition syndrome," *BJOG: An International Journal of Obstetrics and Gynaecology*, vol. 113, no. s3, pp. 118–135, 2006.

[2] R. Clarke, H. W. Ressom, A. Wang et al., "The properties of high-dimensional data spaces: implications for exploring gene and protein expression data," *Nature Reviews Cancer*, vol. 8, no. 1, pp. 37–49, 2008.

[3] P. Mallick and B. Kuster, "Proteomics: a pragmatic perspective," *Nature Biotechnology*, vol. 28, no. 7, pp. 695–709, 2010.

[4] M. L. Bermingham, R. Pong-Wong, A. Spiliopoulou et al., "Application of high-dimensional feature selection: evaluation for genomic prediction in man," *Scientific Reports*, vol. 5, Article ID 10312, 2015.

[5] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society B*, vol. 58, no. 1, pp. 267–288, 1996.

[6] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *The Annals of Statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[7] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, vol. 67, no. 2, pp. 301–320, 2005.

[8] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *The Annals of Statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[9] P. Bühlmann and T. Hothorn, "Boosting algorithms: regularization, prediction and model fitting," *Statistical Science*, vol. 22, no. 4, pp. 477–505, 2007.

[10] N. Meinshausen and P. Bühlmann, "High-dimensional graphs and variable selection with the lasso," *The Annals of Statistics*, vol. 34, no. 3, pp. 1436–1462, 2006.

[11] C. Leng, Y. Lin, and G. Wahba, "A note on the lasso and related procedures in model selection," *Statistica Sinica*, vol. 16, no. 4, pp. 1273–1284, 2006.

[12] N. Meinshausen and P. Bühlmann, "Stability selection," *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, vol. 72, no. 4, pp. 417–473, 2010.

[13] R. D. Shah and R. J. Samworth, "Variable selection with error control: another look at stability selection," *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, vol. 75, no. 1, pp. 55–80, 2013.

[14] B. Hofner, L. Boccuto, and M. Göker, "Controlling false discoveries in high-dimensional situations: boosting with stability selection," *BMC Bioinformatics*, vol. 16, no. 1, article 144, 2015.

[15] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[16] J. Bi, K. P. Bennett, M. Embrechts, C. M. Breneman, and M. Song, "Dimensionality reduction via sparse support vector machines," *Journal of Machine Learning Research*, vol. 3, pp. 1229–1243, 2003.

[17] Y. Wu, D. D. Boos, and L. A. Stefanski, "Controlling variable selection by the addition of pseudovariables," *Journal of the American Statistical Association*, vol. 102, no. 477, pp. 235–243, 2007.

[18] V. G. Tusher, R. Tibshirani, and G. Chu, "Significance analysis of microarrays applied to the ionizing radiation response," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 98, no. 9, pp. 5116–5121, 2001.

[19] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics, Springer New York Inc., New York, NY, USA, 2001.

[20] G. Ridgeway, "The state of boosting," *Computing Science and Statistics*, vol. 31, pp. 172–181, 1999.

[21] A. Mayr, B. Hofner, and M. Schmid, "The importance of knowing when to stop: a sequential stopping rule for component-wise gradient boosting," *Methods of Information in Medicine*, vol. 51, no. 2, pp. 178–186, 2012.

[22] T. Hepp, M. Schmid, O. Gefeller, E. Waldmann, and A. Mayr, "Approaches to regularized regression—a comparison between gradient boosting and the lasso," *Methods of Information in Medicine*, vol. 55, no. 5, pp. 422–430, 2016.

[23] A.-C. Haury, F. Mordelet, P. Vera-Licona, and J.-P. Vert, "TIGRESS: trustful inference of gene regulation using stability selection," *BMC Systems Biology*, vol. 6, article 145, 2012.

[24] S. Ryali, T. Chen, K. Supekar, and V. Menon, "Estimation of functional connectivity in fMRI data using stability selection-based sparse partial correlation with elastic net penalty," *NeuroImage*, vol. 59, no. 4, pp. 3852–3861, 2012.

[25] J. Thomas, A. Mayr, B. Bischl, M. Schmid, A. Smith, and B. Hofner, Stability selection for component-wise gradient boosting in multiple dimensions, 2016.

[26] A. Mayr, B. Hofner, and M. Schmid, "Boosting the discriminatory power of sparse survival models via optimization of the concordance index and stability selection," *BMC Bioinformatics*, vol. 17, no. 1, article 288, 2016.

[27] H. Strasser and C. Weber, "The asymptotic theory of permutation statistics," *Mathematical Methods of Statistics*, vol. 8, no. 2, pp. 220–250, 1999.

[28] M. B. Kursa, A. Jankowski, and W. . Rudnicki, "Boruta—a system for feature selection," *Fundamenta Informaticae*, vol. 101, no. 4, pp. 271–285, 2010.

[29] M. Lang, B. Bischl, and D. Surmann, "batchtools: Tools for R to work on batch systems," *The Journal of Open Source Software*, vol. 2, no. 10, 2017.

[30] U. Alon, N. Barka, D. A. Notterman et al., "Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 96, no. 12, pp. 6745–6750, 1999.

[31] E. Gravier, G. Pierron, A. Vincent-Salomon et al., "A prognostic DNA signature for T1T2 node-negative breast cancer patients," *Genes Chromosomes and Cancer*, vol. 49, no. 12, pp. 1125–1134, 2010.

[32] P. Bühlmann, M. Kalisch, and L. Meier, "High-dimensional statistics with a view toward applications in biology," *Annual Review of Statistics and Its Application*, vol. 1, no. 1, pp. 255–278, 2014.

[33] T. Hothorn, P. Buehlmann, T. Kneib, M. Schmid, and B. Hofner, mboost: Model-Based Boosting. R package version R package version 2.7-0, 2016.

[34] B. Hofner and T. Hothorn, stabs: Stability Selection with Error Control. R package version R package version 0.5-1, 2015.

[35] J. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2010.

[36] G. Tutz and H. Binder, "Generalized additive modeling with implicit variable selection by likelihood-based boosting," *Biometrics*, vol. 62, no. 4, pp. 961–971, 2006.

[37] A. Mayr, H. Binder, O. Gefeller, and M. Schmid, "The evolution of boosting algorithms: From machine learning to statistical modelling," *Methods of Information in Medicine*, vol. 53, no. 6, pp. 419–427, 2014.

[38] A. Mayr, N. Fenske, B. Hofner, T. Kneib, and M. Schmid, "Generalized additive models for location, scale and shape for high dimensional data—a flexible approach based on boosting," *Journal of the Royal Statistical Society. Series C. Applied Statistics*, vol. 61, no. 3, pp. 403–427, 2012.

5. mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions

Declaration of the specific contributions of the author For the publication *mlrMBO: A modular framework for model-based optimization of expensive black-box functions* the software package was mainly written by Bernd Bischl and is maintained by Jakob Richter. Michel Lang, Daniel Horn and Jakob Bossek helped with the implementation. The author mainly conducted and summarized the benchmarks on synthetic test functions and the HPOlib in Section 4. Additional benchmarks for the default settings of the methods were done by the author as well. The description of the software implementation and example usage sections were initially done by the author. The author also contributed to the introduction and motivation of the paper. The text for multi-objective optimization and the text describing the benchmark on multi-objective synthetic test functions have been done by Daniel Horn. He also implemented large parts of the multi-objective optimization code in `mlrMBO`. General text improvements have been done by all authors.

mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions

Bernd Bischl[a], Jakob Richter[b], Jakob Bossek[c], Daniel Horn[b], Janek Thomas[a], Michel Lang[b]

[a] *Ludwig-Maximilians-Universität München, Germany*
[b] *TU Dortmund University, Germany*
[c] *Westfälische-Wilhelms Universität Münster, Germany*

Abstract

We present `mlrMBO`, a flexible and comprehensive `R` toolbox for model-based optimization (MBO), also known as Bayesian optimization, which addresses the problem of expensive black-box optimization by approximating the given objective function through a surrogate regression model. It is designed for both single- and multi-objective optimization with mixed continuous, categorical and conditional parameters. Additional features include multi-point batch proposal, parallelization, visualization, logging and error-handling. `mlrMBO` is implemented in a modular fashion, such that single components can be easily replaced or adapted by the user for specific use cases, e.g., any regression learner from the `mlr` toolbox for machine learning can be used, and infill criteria and infill optimizers are easily exchangeable. We empirically demonstrate that `mlrMBO` provides state-of-the-art performance by comparing it on different benchmark scenarios against a wide range of other optimizers, including DiceOptim, rBayesianOptimization, SPOT, SMAC, Spearmint, and Hyperopt.

Keywords: Model-Based Optimization, Bayesian Optimization, Black-Box Optimization, Hyperparameter Tuning, Parameter Configuration, R

1. Introduction

Black-box functions are systems that require a number of input parameters to produce one or multiple (numeric) outputs. In most cases these are (a) expensive to evaluate in terms of time and/or monetary cost, and (b) knowledge of their internal working is not available, which often manifests through the absence of derivatives. Such problems occur in production engineering [e.g. 1], where the inputs are possible settings of industrial machines or used materials

Email addresses: `bernd_bischl@gmx.net` (Bernd Bischl), `jakob.richter@tu-dortmund.de` (Jakob Richter), `bossek@wi.uni-muenster.de` (Jakob Bossek), `daniel.horn@tu-dortmund.de` (Daniel Horn), `janek.thomas@stat.uni-muenchen.de` (Janek Thomas), `lang@statistik.tu-dortmund.de` (Michel Lang)

and the output is one or multiple measurements regarding the quality of fabricated parts. Since this makes a single evaluation expensive, one tries to find the optimal settings of production steps in a minimal number of tries. Design of Computer Experiments (DACE) [2] is a discipline focused on solving such problems and sequential model-based optimization (SMBO) [3] has become the state-of-the-art optimization strategy in recent years.

The generic SMBO procedure starts with an initial design of evaluation points, and then iterates the following steps:

1. Fit a regression model to the outcomes and design points obtained so far,
2. query the model to propose a new, promising point, often by optimizing a so-called infill criterion or acquisition function,
3. evaluate the new point with the black-box function and add it to the design.

Several adaptations and extensions, e.g., multi-objective optimization [4], multi-point proposal [5, 6], more flexible regression models [7] or alternative ways to calculate the infill criterion [8] have been investigated recently.

A different field of application for SMBO is the hyperparameter optimization for machine learning methods [e.g. 9, 10, 11]. Here, the black-box is a machine learning method and the objective(s) is one or multiple performance measure(s), validated via resampling on a data set of interest. The black-box function can be more complex, for example a machine learning pipeline which includes preprocessing, feature selection and model selection.

After a brief comparison with related software in Subsection 1.1 and clarification of our main contributions in Subsection 1.2, we introduce the general SMBO procedure in more detail in Section 2. Section 3 highlights the capabilities of our software `mlrMBO`, showcased by some code examples. In Section 4 we empirically demonstrate that `mlrMBO` achieves state-of-the-art performance on a wide range of synthetic and real-world single- and multi-objective scenarios. Section 5 gives an outlook on future work.

1.1. Related Software

We will briefly present an overview of available software for model-based optimization, starting with implementations based on the Efficient Global Optimization algorithm (EGO), i.e., the SMBO algorithm proposed by Jones et al. [3] using Gaussian processes (GPs), and continue with extensions and alternative approaches.

Both `DiceOptim` [12] and `rBayesianOptimization` [13] are `R` packages that offer EGO implementations. A sophisticated EGO implementation can be found in the Python package `Spearmint` [14]. It focuses on hyperparameter optimization of machine learning algorithms with enhancements regarding variable costs of experiments and parallelization. All three packages offer different GP kernels and infill criteria, but only support numerical (non-conditional) parameters

and, except for Spearmint, no multi-criteria optimization or parallelization is available. A multi-criteria version of Spearmint is introduced in [15].

The C++ library `BayesOpt` [16] contains an extended version of EGO, including Student-t processes, support of mixed and conditional parameters as well as meta-criteria algorithms to automatically find reasonable infill criteria during optimization. It offers interfaces for Python, Matlab and Octave.

`SMAC` [7] is one of the most established frameworks and allows to optimize mixed parameter spaces as it uses a random forest instead of a GP for regression. Besides general black-box optimization, it is focused on algorithm configuration. However, `SMAC` is limited to single-criteria optimization and parallelization is not supported.

`Hyperopt` [8] is an optimization package in Python that supports numerical, categorical and conditional parameters. Instead of a regression it uses a tree of Parzen estimators (TPE) to compute point suggestions. It supports distributed parallel and asynchronous execution. `Hyperopt` can be used for general black-box optimization, but is mainly focused on machine learning tasks.

Another `R` implementation for sequential black-box optimization is `SPOT` [17]. It is a toolbox with different modeling techniques and offers a wide variety of statistical methods. `SPOT` contains sophisticated algorithms to handle functions with noisy evaluations, is able to handle constrains in functions and supports multi-objective optimization.

1.2. Main Contributions and Prior Applications

The main contribution of this paper is the presentation of the `R` package `mlrMBO`, which implements a generic SMBO framework and provides a large variety of different SMBO methods due to its modular structure. `mlrMBO` is even more flexible than SPOT in its choice of surrogate models as it is connected to the `R` package `mlr` [18] which interfaces more than 60 machine learning regression algorithms. Besides the default SMBO procedure, `mlrMBO` focuses on three domains: Mixed parameter space optimization, multi-point proposals and multi-objective optimization. Even combinations of the three domains are possible, which to our knowledge no other software is currently capable of. `mlrMBO` is easy to use as many default implementations for the individual steps of the SMBO procedure are directly supported in a plug-and-play style. Simple interfaces are available to extend the package with user specific variants.

Benchmarks show that `mlrMBO` achieves state-of-the-art performance in each domain. Additionally, `mlrMBO` has been successfully applied in some practical settings. In [19, 20] it was used to optimize the hyperparameters of machine learning pipelines (joint pre-processing and model hyperparameters) for support vector machines and general machine learning models, respectively, in a single objective setting. Hess et al. [21] proposed an `mlrMBO` ensemble-based approach to identify the best surrogate model during optimization through reinforcement learning. Horn et al. [22] considered a multi-objective benchmark and optimized the runtime-accuracy trade-offs of several approximate support vector machine solvers. Horn and Bischl [11] introduced the general capability of `mlrMBO` to

solve multi-objective machine learning tasks. Steponavičė et al. [23] investigated the impact of different initial design sampling techniques on the performance of multi-objective model-based optimization methods by using `mlrMBO`.

2. Sequential Model-Based Optimization

This section describes the general SMBO setup and presents the individual building blocks in Subsection 2.1. While SMBO is modular and can thus be customized for a variety of different tasks, we highlight the most prominent combinations of components described in the literature like EGO [3] (Subsection 2.2) or SMAC-like [7] optimizers (Subsection 2.3). Subsections 2.4 and 2.6 introduce parallelization through multi-point proposal, and multi-objective optimization.

2.1. Sequential model-based optimization

Let $f(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}$ be an arbitrary black-box function with a d-dimensional input domain $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_d$ and a deterministic output y. Each $\mathcal{X}_i$ ($i = 1, \cdots, d$) can be either numeric and bounded (i.e. $\mathcal{X}_i = [l_i, u_i] \subset \mathbb{R}$) or a finite set of s categorical values ($\mathcal{X}_i = \{v_{i1}, \ldots, v_{is}\}$). Without loss of generality, we want to find the input $\boldsymbol{x}^*$ with

$$\boldsymbol{x}^* = \underset{\boldsymbol{x} \in \mathcal{X}}{\arg\min}\, f(\boldsymbol{x}).$$

In the context of model-based optimization, we usually assume that f is expensive to evaluate, hence the total number of function evaluations is limited by a budget. At the heart of SMBO are so-called surrogate models $\hat{f}$ which cheaply estimate the expensive black-box function f and which are iteratively updated and refined. The general approach is illustrated in Figure 1. The figure outlines the following steps, whereas each step is explained in more detail in the following subsections:

(1) An initial design of n_{init} points $\boldsymbol{x}^{(j)}$ ($j = 1, \ldots, n_{\text{init}}$) is sampled from $\mathcal{X}$ and f is evaluated at these points to yield outcomes $y^{(j)} = f(\boldsymbol{x}^{(j)})$. The tuples $\left(y^{(j)}, \boldsymbol{x}^{(j)}\right)$ constitute the data to build the initial surrogate model $\hat{f}$ in the next step.

(2) Fit a surrogate model to all evaluated points $\boldsymbol{x}^{(j)} \in \mathcal{X}$ and corresponding values $y^{(j)}$.

(3) An *infill criterion* proposes m points $\boldsymbol{x}^{(j+i)}$ ($i = 1, \ldots, m$). The criterion is defined on $\mathcal{X}$ and operates on the surrogate $\hat{f}$ to determine points which are promising for the optimization. These points should either have a good expected objective value or high potential to improve the quality of the surrogate model.

(4) The proposed points are evaluated using f and the new tuples $\left(y^{(j+i)}, \boldsymbol{x}^{(j+i)}\right)$ are added to the design.

(5) If the budget is not exhausted (and no other termination criteria is met), go to step (2).

(6) If the budget is exhausted or another termination criteria is met, return the proposed solution for the optimization problem.

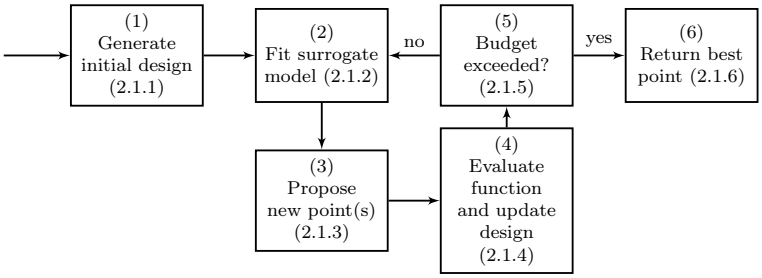

Figure 1: General SMBO approach.

2.1.1. Initial Design

The initial design specifies the points of the input domain at which the black-box function is evaluated to build the initial surrogate model $\hat{f}$. If too few points are chosen or if the points do not cover $\mathcal{X}$ well, the fit of $\hat{f}$ may be poor and thus points proposed based on $\hat{f}$ may be suboptimal for the progress of the optimization. Fitting a surrogate model may even be impossible. On the other hand, a large initial design may reduce the available budget too much. `mlrMBO` provides various options for the initial design: The user can specify it manually or generate designs either completely at random, coarse grid designs or by using space-filling Latin Hypercube Designs [24].

2.1.2. Surrogate Models

One of the main factors that determines the choice of surrogate model $\hat{f}$ is the structure of the input space $\mathcal{X}$. If $\mathcal{X} \subset \mathbb{R}^d$, *Kriging* [3] is the recommended choice and provides state-of-the-art performance. In Section 2.2, the Kriging-based EGO approach is discussed in more detail. If the search space $\mathcal{X}$ also includes categorical parameters on the other hand, *random forests* are a viable alternative [7] as they can handle such parameters directly, without the need to encode the categorical parameters as numeric. `mlrMBO` allows the use of any of the many regression models available in the `R` package `mlr`, which itself can also be easily extended to support custom regression learners [25].

While Kriging models and random forests already provide uncertainty estimation natively, generic bagging can be applied to arbitrary regression models to retrieve standard error estimators in `mlr`.

2.1.3. Infill Criteria

The infill criterion, or sometimes called acquisition function, guides the optimization and tries to trade-off exploitation and exploration. This is usually achieved by combining $\hat{\mu}(\boldsymbol{x})$ and $\hat{s}(\boldsymbol{x})$ (or $\hat{s}^2(\boldsymbol{x})$) in a single formula in a well-balanced fashion, where the posterior mean $\hat{\mu}(\boldsymbol{x})$ and posterior standard deviation $\hat{s}(\boldsymbol{x})$ (or posterior variance $\hat{s}^2(\boldsymbol{x})$) are estimated by the surrogate model $\hat{f}$. $\hat{s}(\boldsymbol{x})$ and $\hat{s}^2(\boldsymbol{x})$ are sometimes also called "local uncertainty estimators". Assuming that our model $\hat{f}$ is somewhat "spatial" in the sense that higher values of $\hat{s}(\boldsymbol{x})$ indicate regions of the search space that few of our design points lie close to and / or we have not learned the structure of f very well at $\boldsymbol{x}$, we are therefore looking for points with low $\hat{\mu}(\boldsymbol{x})$ and high $\hat{s}(\boldsymbol{x})$.

Arguably the most popular choice is the *expected improvement*

$$\mathrm{EI}(\boldsymbol{x}) := \mathrm{E}(I(\boldsymbol{x}))$$

where the random variable $I(\boldsymbol{x})$ defines the potential improvement at $\boldsymbol{x}$ over the currently best observed function value $y_{\min}$:

$$I(\boldsymbol{x}) := \max\left\{y_{\min} - Y(\boldsymbol{x}), 0\right\}.$$

Here, $Y(\boldsymbol{x})$ is a random variable that should express the posterior distribution at $\boldsymbol{x}$, estimated with $\hat{f}$. For a Gaussian process, $Y(\boldsymbol{x})$ is normally distributed with $Y(\boldsymbol{x}) \sim N(\hat{\mu}(\boldsymbol{x}), \hat{s}^2(\boldsymbol{x}))$. Under this assumption, $\mathrm{EI}(\boldsymbol{x})$ can be expressed analytically in closed form as

$$\mathrm{EI}(\boldsymbol{x}) = (y_{\min} - \hat{\mu}(\boldsymbol{x}))\,\Phi\left(\frac{y_{\min} - \hat{\mu}(\boldsymbol{x}))}{\hat{s}(\boldsymbol{x})}\right) + \hat{s}(\boldsymbol{x})\phi\left(\frac{y_{\min} - \hat{\mu}(\boldsymbol{x})}{\hat{s}(\boldsymbol{x})}\right), \quad (1)$$

where Φ and ϕ are the distribution and density function of the standard normal distribution, respectively.

A simpler approach to balance $\hat{\mu}(\boldsymbol{x})$ and $\hat{s}(\boldsymbol{x})$ for a point $\boldsymbol{x}$ is given by the *lower confidence bound*

$$\mathrm{LCB}(\boldsymbol{x}, \lambda) = \hat{\mu}(\boldsymbol{x}) - \lambda\hat{s}(\boldsymbol{x}), \quad (2)$$

where $\lambda > 0$ is a constant that controls the "mean vs. uncertainty" trade-off.

Furthermore, `mlrMBO` currently support pure mean $\hat{\mu}(\boldsymbol{x})$ minimization (pure exploitation) and pure uncertainty $\hat{s}(\boldsymbol{x})$ maximization (pure exploration) and further criteria for multiple point proposals (see Section 2.4), noisy optimization (see Section 2.5), and multi-objective optimization (See section 2.6).

2.1.4. Infill Optimization

The infill optimizer searches for the point $\boldsymbol{x}$ which yields the best infill value. Unlike the original optimization problem on f, the optimization on the infill criterion can be considered inexpensive. While this is still a black-box optimization problem, points can be evaluated more lavishly, and Jones et al. [3] propose a branch and bound algorithm for this task. `mlrMBO` defaults to a more generic approach, which we call *focus search*, outlined in Algorithm 1. It is able to handle

numeric parameter spaces, categorical parameter spaces, as well as mixed and hierarchical spaces. The algorithm starts with a large random design from which all points are evaluated by the surrogate regression model to determine the most promising point. Next, focus search shrinks the search space around the best point and samples new random points for the now focused search space. The shrinkage of search space is iterated n_{iters} times. The complete procedure can be restarted n_{restart} times to avoid local optima. Finally the best point over all restarts and iterations is returned. Evolutionary algorithms like CMA-ES [26] or custom user-defined optimizers can be selected alternatively.

Algorithm 1 Infill Optimization: Focus Search.

Require: infill criterion $c : \mathcal{X} \to \mathbb{R}$, control parameters n_{restart}, n_{iters}, n_{points}

for $u \in \{1, ..., n_{\text{restart}}\}$ **do**
  Set $\tilde{\mathcal{X}} = \mathcal{X}$
  for $v \in \{1, ..., n_{\text{iters}}\}$ **do**
    generate random design $\mathcal{D} \subset \tilde{\mathcal{X}}$ of size n_{points}
    compute $\boldsymbol{x}^*_{u,v} = (x_1^*, ..., x_d^*) = \arg\min_{\boldsymbol{x} \in \mathcal{D}} c(\boldsymbol{x})$
    shrink $\tilde{\mathcal{X}}$ by focusing on $\mathbf{x}^*$:
    for each search space dimension $\tilde{\mathcal{X}}_i$ in $\tilde{\mathcal{X}}$ **do**
      if $\tilde{\mathcal{X}}_i$ numeric: $\tilde{\mathcal{X}}_i = [l_i, u_i]$ **then**
        $l_i = \max\{l_i, x_i^* - \frac{1}{4}(u_i - l_i)\}$
        $u_i = \min\{u_i, x_i^* + \frac{1}{4}(u_i - l_i)\}$
      end if
      if $\tilde{\mathcal{X}}_i$ categorical: $\tilde{\mathcal{X}}_i = \{v_{i1}, \ldots, v_{is}\}$, $s > 2$ **then**
        $\bar{x}_i$ = sample one category uniformly from $\tilde{\mathcal{X}}_i \backslash x_i^*$
        $\tilde{\mathcal{X}}_i = \tilde{\mathcal{X}}_i \backslash \bar{x}_i$
      end if
    end for
  end for
end for
Return $\boldsymbol{x}^* = \underset{u \in \{1,...,n_{\text{restart}}\}, v \in \{1,...,n_{\text{iters}}\}}{\arg\min} c(\boldsymbol{x}^*_{u,v})$

2.1.5. Termination

Multiple termination criteria can be used in `mlrMBO`. Commonly a limit is set for the total number of evaluations of f or for the number of SMBO iterations. Alternatively, the optimization can be terminated after a given time or after a time budget for function evaluations is exhausted. The optimization can also be stopped as soon as a predefined objective value is reached. Furthermore, the user can create custom termination rules.

2.1.6. Final Point

Finally, the final solution $\boldsymbol{x}^*$ has to be determined. Usually the best point observed during the optimization is picked. Fitting a last surrogate model to find the best point predicted is a viable option, especially if f is noisy.

2.2. Efficient Global Optimization (EGO)

Kriging models [27] are arguable the most popular choice for a surrogate model because they are very flexible and provide a local uncertainty estimator [3].

In general, we consider a numeric-only input domain $\mathcal{X} \subset \mathbb{R}^d$. Jones et al. [3] were the first who introduced surrogate models for the sequential optimization of box-constrained functions with real-valued arguments. Their Efficient Global Optimization (EGO) algorithm employs Kriging models together with the expected improvement infill criterion (see Equation 1). Maximizing the EI results in an infill criterion that balances exploitation of the model structure and exploration of regions with high uncertainty and has proven to be highly effective [3]. It can ensure global convergence [28, 29] (which is somewhat unrealistic to expect under the usually tight budget constraints that exist for many expensive black-box optimization problems).

Figure 2 illustrates the point proposal at the 3rd (left) and the 4th iteration (right) of an EGO run on a $1d$ cosine mixture function. It illustrates how high uncertainty ($\hat{s}$) and a low value of $\hat{\mu}$ contribute to the EI and thus to the selection of the next point and the ability of model-based optimization to find the optimum even for multi-modal functions.

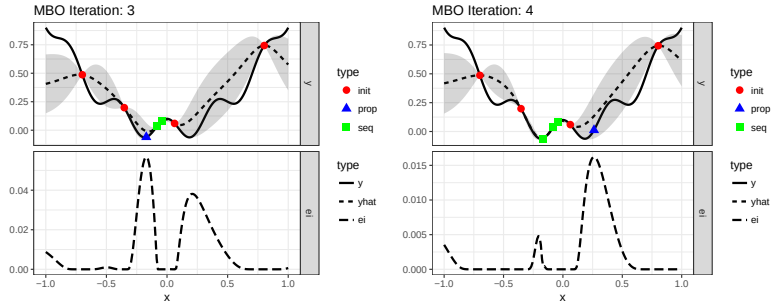

Figure 2: State at the 3rd (left) and 4th iteration (right) of an exemplary EGO run on a $1d$ cosine mixture function. The upper part shows the real function f as a solid line and its estimation $\hat{\mu}$ dotted. The uncertainty is indicated by the shaded area. Initial design points are displayed as red circles, sequential points as green squares. The lower part shows the respective value for the EI. The optimum of the EI defines the point that proposed to be evaluated next (blue triangle).

2.3. Mixed Space Optimization

Real life scenarios often include mixed-valued as well as hierarchical parameter spaces with conditional parameters. An example is the tuning of a support vector machine, for which the parameter space is illustrated in Figure 3. Depending on the choice of the *kernel*, the hyperparameter γ has to be optimized for the *radial* kernel (so it is conditional on the setting of *kernel*), but it is not present (or we could say: active) for the *linear* kernel. In contrast to γ, the

hyperparameter C is unconditionally always active. Kriging is not really suited for such problem domains, since covariance kernels natively supporting those types of data are still subject to research [30].

For the *initial design* all options support categorical parameters as well as hierarchical dependencies (feasible values of a parameter depend on the values of other parameters).

For the *surrogate* we need a regression model that is more flexible and can handle categorical features as well as missing values to support dependent parameters. A slightly modified *random forest* can be used for this purpose. If a hyperparameter is not active in a design point in the training set (due to unfulfilled conditions), we will mark its value as missing. Although the random forest could potentially directly handle missing values, many implementations do not. Hence, we impute these values in the following way: For categorical parameters we code missing values as a new level, and for numerical parameters we code the imputed value out of the range of the box-constraints of the parameter under consideration. This is known as the *separate-class method* and was shown to perform best for decision trees in a prediction-oriented study, when missingness is related to the outcome [31].

In order to still use *infill criteria* as LCB and EI, we also have to compute an uncertainty estimate $\hat{s}(\boldsymbol{x})$ for the random forest. For bagging-like predictors this can be computed or approximated in various ways from the bootstrap. We refer the reader to [32, 33] for further details. In `mlr` the uncertainty estimator can be deviated from an expensive extra bootstrap around the random forest, the jackknife, the infinitesimal jackknife, or a simple estimator which extracts the standard error simply from the internal bootstrap of the random forest. In our experience, the jackknife estimator works most reliably, so it is the current default for `mlrMBO` with random forests as surrogate. However, it should be noted that the random forest is not really a spatial model as a Gaussian process and therefore the properties of the uncertainty estimator are less intuitive in comparison to the ones from Kriging models. Our following results still indicate that we obtained state-of-the-art results with this default, and we deem this aspect a matter for further research.

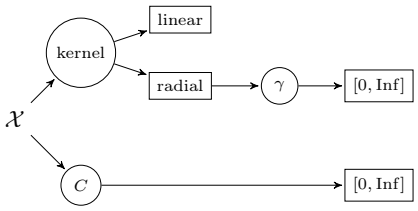

Figure 3: Dependent search space for the tuning of a support vector machine. Circles denote parameters, rectangles denote parameter ranges, arrows denote the hierarchical structure.

2.4. Multi-Point Proposal

The expensive nature of the optimization problem makes parallelization, i.e. the evaluation of different configurations on multiple CPUs, an important extension to speed up the SMBO process. Recently many methods have been proposed to simultaneously propose m points in each iteration. We showcase three methods implemented in `mlrMBO`, which are also discussed in [6]. A straightforward approach is *qLCB* [34], an extension of the LCB criterion. Instead of one fixed λ, multiple λ_k $(k = 1, \ldots, m)$ are drawn from an exponential distribution to obtain m points $\boldsymbol{x}^{(j+1)}, \ldots, \boldsymbol{x}^{(j+m)}$:

$$\text{qLCB}(\boldsymbol{x}, \lambda_k) = \hat{\boldsymbol{y}}(\boldsymbol{x}) - \lambda_k \hat{s}(\boldsymbol{x}), \quad \lambda_k \sim \text{Exp}\left(\frac{1}{\lambda}\right).$$

The criterion is than optimized separately for every λ_k, so that overall m points are proposed. Proposals that were obtained by optimizing the qLCB for a low value of λ_k exploit the model and are in proximity of the best found y so far. For high values of λ_k the proposals will be of exploratory nature. This ensures that in one SMBO iteration all proposals balance exploitation and exploration.

Another approach to propose multiple points using the expected improvement is known as *constant liar* [5]. Here we obtain $\boldsymbol{x}^{(j+1)}$ in the same way as for the ordinary EI. To obtain $\boldsymbol{x}^{(j+2)}$ we assume that the evaluation at point $\boldsymbol{x}^{(j+1)}$ is done and update the surrogate model with a made up target value y. Exemplary choices for the made up value are $\min(\boldsymbol{y})$, $\max(\boldsymbol{y})$, the mean $\bar{\boldsymbol{y}}$, or the predicted posterior mean $\hat{\mu}(\boldsymbol{x}^{(j+1)})$ of the surrogate model. The latter approach is also often referred to as *kriging believer*.

Bischl et al. [6] propose the multi-objective infill model-based optimization (MOIMBO) approach. The posterior mean $\hat{\mu}(\boldsymbol{x})$ and variance $\hat{s}(\boldsymbol{x})$ are not scalarized in a single function (as done by EI or (q)LCB), instead a multi-objective optimization strategy (see Section 2.6) is used to optimize them jointly and propose a whole set of optimal points. To ensure that the points are diverse, a distance measure, e.g. the nearest neighbor distance, can be used as a third objective.

2.5. Noisy Optimization

Noisy optimization assumes that the objective function f is stochastic. Usually, one now faces the problem to optimize $\text{E}[f(x)]$ instead of $f(x)$ and common strategies are intelligent repetition strategies [35] or adapted infill criteria. `mlrMBO` currently only offers the latter (but of course the user can always opt to perform averaging in the objective function, e.g. by naively averaging over a constant number of repetitions himself).

A popular infill criterion for noisy functions is the *expected quantile improvement* [36] which is an extension of EI. Instead of looking for an improvement over best value observed so far (the $y_{\min}$ in the EI formula), we exchange this with a so called "plug-in" value $q_{\min}$:

$$\text{EQI}(\boldsymbol{x}) = (q_{\min} - q(\boldsymbol{x}))\, \Phi\left(\frac{q_{\min} - q(\boldsymbol{x})}{s_{q(\boldsymbol{x})}}\right) + s_{q(\boldsymbol{x})} \phi\left(q_{\min} - q(\boldsymbol{x}) s_{q(\boldsymbol{x})}\right), \quad (3)$$

where $q_{\min}$ is the lowest β-quantile $q(\boldsymbol{x}_i)$ for all previously evaluated points $\boldsymbol{x} \in \{\boldsymbol{x}^1, \dots \boldsymbol{x}^n\}$, and β is a user control parameter for the EQI. The estimated β-quantile at point $\boldsymbol{x}$ is given by $q(\boldsymbol{x}) = \hat{\mu}(\boldsymbol{x}) + \Phi^{-1}(\beta)\hat{s}(\boldsymbol{x})$. This implies that the criterion will be non-zero at already evaluated points allowing re-evaluations or evaluations very close to already evaluated design points to increase knowledge of promising points.

`mlrMBO` offers also the so called "augmented expected improvement" and its modular design makes extensions towards further criteria functions straightforward. For a further in-depth discussion of this topic we refer the reader to [37] and their benchmark for noisy MBO approaches.

2.6. Model-Based Multi-Objective (MBMO) Optimization

Multi-objective optimization problems are characterized by a set of target functions $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \dots, f_k(\boldsymbol{x}))$ which have to be optimized simultaneously. Since there is no total order in $\mathbb{R}^k$, for $k \geq 2$, the concept of *Pareto dominance* is used. A point $\boldsymbol{x}$ pareto-dominates another point $\tilde{\boldsymbol{x}}$, $\boldsymbol{x} \preceq \tilde{\boldsymbol{x}}$, if $f_i(\boldsymbol{x}) \leq f_i(\tilde{\boldsymbol{x}})$ for $i = 1, \dots, k$ and $\exists\, j\, f_j(\boldsymbol{x}) < f_j(\tilde{\boldsymbol{x}})$, i.e., $\boldsymbol{x}$ needs to be as good as $\tilde{\boldsymbol{x}}$ in each component and strictly better in at least one. A point $\boldsymbol{x}$ is said to be *non-dominated* if it is not dominated by any other point. The set $P = \{\boldsymbol{x} \mid \nexists\, \tilde{\boldsymbol{x}}\ \tilde{\boldsymbol{x}} \preceq \boldsymbol{x}\}$ of all non-dominated points is called the *Pareto set*. It contains all incomparable trade-off solutions. In multi-objective optimization the goal is to approximate the Pareto set or the *Pareto front* $f(P)$, i.e., the image of P under f.

In recent years some approaches were published that generalize single-objective SMBO algorithms like EGO for the multi-objective case. We distinguish between 3 different MBMO algorithm classes: First, *scalarization based algorithms* that use EGO to optimize a scalarized version of the black-box functions with random weights for the scalarization in each iteration. Second, *Pareto based algorithms* that fit individual models for each objective and perform multi-objective optimization of infill criteria on these models. Third, *direct indicator based algorithms* that also fit individual models, but perform a single objective optimization of an infill criterion aggregating all models. `mlrMBO` supports 4 different MBMO algorithms, covering all 3 classes: ParEGO [38] as scalarization based, MSPOT [39] as Pareto based, and both SMS-EGO [40] and ε-EGO [41] as direct indicator based algorithms.

A much more detailed discussion of these methods, their multi-point variants, and what is currently implemented in `mlrMBO` is given in [4].

3. mlrMBO R Package

We implemented the software package `mlrMBO` for the statistical programming language `R`. It is designed as a modular framework. The individual components of model-based optimization such as the infill criterion or the stopping conditions (cf. Section 2) can easily be combined in a plug-and-play fashion to respect the specific characteristics of the optimization problem at hand. In the following we give a short introduction of this process which is split into multiple steps.

Definition of the black-box function. For the first step `mlrMBO` relies on the `smoof` package [42] which provides a unified interface to work with black-box functions. Many test functions that are frequently used to benchmark optimizers are already included. Additionally, the package provides the functions `make{Single,Multi}ObjectiveFunction()` as constructors for custom test functions. Mandatory arguments are the function itself, a name and a parameter set. In the simplest case, the latter is defined by names and box constraints, which can be specified concisely using the `ParamHelpers` package. For more complex settings, it is also possible to connect parameters with arbitrary transformation functions (e.g., to vary a parameter on the log-scale) or declare dependencies between parameters. The following listing gives an example for the definition of the black-box $f(\mathbf{x}) = (x_2 - 0.1x_1^2 + x_1 - 6)^2 + \cos(x_1)$ with $x_1 \in [-5, 10], x_2 \in [0, 15]$:

```
fn = makeSingleObjectiveFunction(
  name = "my_blackbox",
  fn = function(x) (x[2] - 0.1 * x[1]^2 + x[1] - 6)^2 + cos(x[1]),
  par.set = makeParamSet(
    makeNumericParam("x1", lower = -5, upper = 10),
    makeNumericParam("x2", lower = 0, upper = 15)
  )
)
```

Definition of the Initial Design. To specify the points to be evaluated to initialize the surrogate an initial design has to be specified. It is recommended to use a Latin Hypercube Design by calling `generateDesign()` and passing the number of desired points. If no design is given by the user, `mlrMBO` will generate a *maximin* Latin Hypercube Design of size 4 times the number of the black-box function's parameters.

Definition of the surrogate regression model. `mlrMBO` builds up on the `mlr` package [18], which offers a unified interface for a plethora of machine learning methods in `R`. For surrogate regression, Kriging (`makeLearner("regr.km")`) and random forests (`makeLearner("regr.randomForest")`) are popular choices, but other regression methods can be selected as well. Keep in mind that if expected improvement or LCB is chosen as the infill criterion, the surrogate either has to provide an uncertainty estimator, or has to be combined with a bagging approach using the `makeBaggingWrapper()` in `mlr`. If no regression method is supplied by the user, the fallback is a Kriging model with a Matern-$3/2$ kernel and the "GENetic Optimization Using Derivatives" (genoud) fitting algorithm in a fully numeric setting, and a random forest with jackknife variance estimation otherwise.

Definition of the control flow. Basic settings like the number of proposed points in each SMBO iteration or the error handling are set via `makeMBOControl()`

which returns a base control object. This object can be further extended to adjust the different component of the SMBO methodology. `setMBOControlInfill()` adjusts the infill criterion and the infill criterion optimizer. If the infill optimization is unspecified, `mlrMBO` uses LCB as infill criterion with $\lambda = 1$ in a fully numeric setting and $\lambda = 2$ if at least one one discrete parameter is present. To optimize the criterion, focus search with $n_{\text{restarts}} = 3$, $n_{\text{iters}} = 5$ and $n_{\text{points}} = 1000$ is used by default. For multi-point proposals or multi-objective optimization, `setMBOControlMultiPoint()` and `setMBOControlMultiObj()` are used, respectively. If multiple points are proposed, they can be evaluated simultaneously using different parallelization (i.e. multicore, sockets, and MPI) and high-performance computation systems (e.g., Slurm, LSF, OpenLava, TORQUE, or Docker Swarms) with the `R` packages `parallelMap` and `batchtools` [43]. Finally, `setMBOControlTermination()` controls the termination criteria.

Putting it all together. The actual optimization is finally started by calling the `mbo()` function with the (optional) initial design, the black-box function, the (optional) surrogate regression method, and the control object as arguments. The following listing demonstrates an application of `mlrMBO` to optimize our example black-box.

```
library(mlrMBO)

# Create initial random Latin Hypercube Design of 10 points
library(lhs) # for randomLHS
des = generateDesign(n = 5L * 2L, getParamSet(fn), fun = randomLHS)

# Specify kriging model with standard error estimation
surrogate = makeLearner("regr.km", predict.type = "se",
  covtype = "matern3_2")

# Set general controls
ctrl = makeMBOControl()
ctrl = setMBOControlTermination(ctrl, iters = 30L)
ctrl = setMBOControlInfill(ctrl, crit = makeMBOInfillCritEI())

# Start optimization
mbo(fn, des, surrogate, ctrl)
```

The resulting object contains the full optimization path, with all x and y values, runtime of function evaluations, final state, potential error messages as well as optionally all fitted surrogate models. Diagnostic visualizations of the optimization are available by calling `plot()` and for one and two dimensional input domains with single- or multi-objective targets, each step of the optimization process can be visualized by calling `exampleRun()` or `exampleRunMultiObj()`. For instance, Figure 2 has been created with `exampleRun()` and `plotExampleRun()`.

4. Benchmarks

In this section, the performance of `mlrMBO` is evaluated on three extensive benchmarks. First, we compare `mlrMBO` against other black-box optimizers connected to `R` (Section 4.1), then against state-of-the-art optimizers that are not available in `R` through the optimization benchmark framework HPOlib [44] (Section 4.2). Furthermore, we summarize a previously published simulation study of multi-objective optimization using `mlrMBO` [4] (Section 4.3). All benchmarks were conducted using the `batchtools` [43] package for `R`.

4.1. Model-Based Single-Objective Optimization

We run our implementation on various single-objective optimization tasks and compare it with the three EGO implementations available in `R`: `DiceOptim` [12], `rBayesianOptimization` [13] and `SPOT` [17]. Additionally, to ensure that an EGO approach is suitable, we also consider a basic random search as well as the popular covariance matrix adaptation evolution strategy (CMA-ES) based on the `R` package `cmaesr` [26].

Benchmarks. The methods are evaluated on a set of six 5-dimensional, continuous, and single-objective test functions: *Alpine01*, *Deflected Corrugated Spring*, *Schwefel*, *Ackley*, *Griewank* and *Rosenbrock*. All are defined in the `R`-package `smoof` and have been subject to optimization benchmarks previously.

Setup. For the initial design, the same pre-generated maximin Latin Hypercube design containing 25 points is used for `mlrMBO`, `DiceOptim`, `SPOT` and the random search. It was not possible to pass a user-defined initial design in `rBayesianOptimization` without provoking an error. Instead, a random design of the same size is generated internally. We allow each algorithm 200 sequential iterations. Since CMA-ES as an evolutionary algorithm does not initialize with a design, it gets an additional budget of 25 iterations (in total 225). All algorithms are run in their default settings carefully chosen by the respective package authors.

Evaluation. The objective values of the proposed solutions are summarized in Figure 4. All methods performed clearly better than the baseline random search approach on all six test functions. In comparison with the other EGO-based algorithms, `mlrMBO` yields a substantial better objective on four test functions and similar objective on the other two. `SPOT` is slightly better than `mlrMBO` on *Griewank*, but worse on three others. The evolutionary CMA-ES is comparable to `mlrMBO` on *Alpine01* and slightly better on *Rosenbrock*, but considerably worse on the four other problems. If we consider the averaged rank of the methods over all test functions as shown in Table 1, `mlrMBO` proves to be the best method overall, with `SPOT` in second place.

Besides the quality of the solution, runtime, and computational overhead should also be considered. The timings for a complete optimization run in

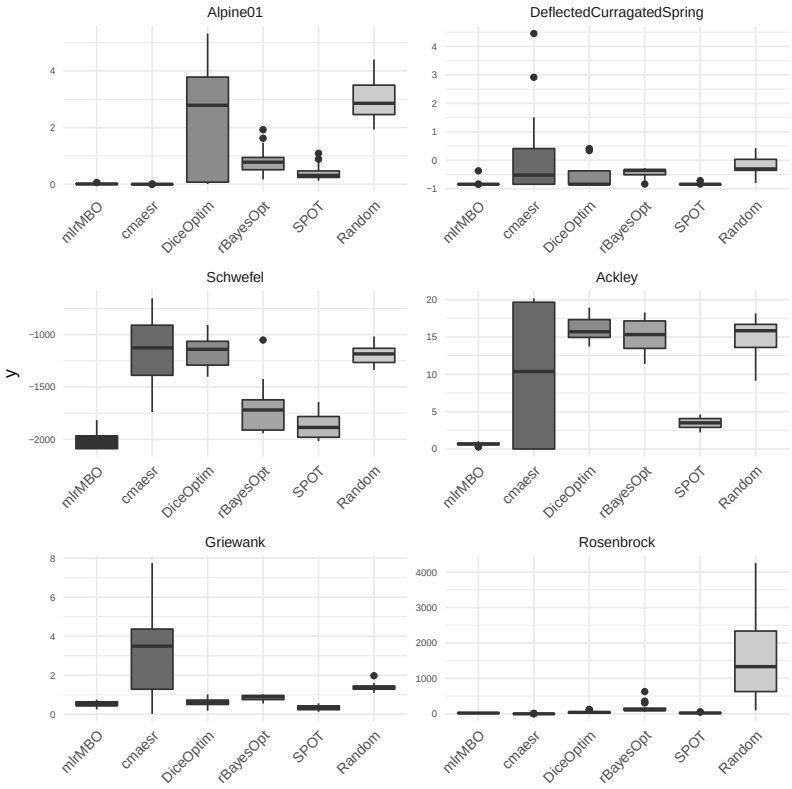

Figure 4: Best objective value (on y axis) found by respective algorithms on respective test function.

minutes are listed in Table 1. Note that we are basically measuring the overhead of the optimization algorithms, as the synthetic test functions are evaluated in microseconds. The random search unsurprisingly comes with the least overhead, followed by CMA-ES as implemented in the package `cmaesr`. The EGO-based approaches consume considerably more time by fitting the surrogate model and optimizing the infill criterion. Here, `mlrMBO` is slower than `DiceOptim` but still more than twice as fast as `SPOT` and orders of magnitudes faster than `rBayesianOptimization`. However, keep in mind that EGO is tailored for expensive problems. If we paid each function evaluation with just one minute of computation time, the differences between 200 min for random search and 212 min for `mlrMBO` seems to be a reasonable price to pay for a much better objective value.

Algorithm	Average Rank	Average runtime in minutes
`mlrMBO`	1.95	8.03
`SPOT`	2.48	27.88
`cmaesr`	3.17	0.01
`DiceOptim`	3.97	3.35
`rBayesOpt`	4.24	695.96
`Random`	5.19	0.00

Table 1: Average ranks and runtime on artificial test functions.

4.2. Model-Based Single-Objective Optimization in Mixed Spaces

The second benchmark compares `mlrMBO` to three[1] other state-of-the-art Bayesian optimizers which are not connected to `R`: Spearmint [14], hyperopt (called TPE in the following) [8] and SMAC [7]. We use the hyperparameter optimization library HPOlib [44], which contains a large number of standardized benchmarks. Besides purely numerical problems, the HPOlib also defines problems with mixed and dependent parameter spaces. We evaluate the methods on four synthetic functions (*branin*, *camelback*, *michalewicz* and *har6*), three parameter optimization problems on grids (linear discriminant analysis (*lda*), logistic regression (*logreg*) and a support vector machine (*svm*)), as well as a deep neural network (*hpnnet*) with 15 parameters, and a deep belief network (*hpdbnet*) with 35 parameters. The latter two problems were originally proposed by Bergstra et al. [8]. `mlrMBO` uses its default settings, i.e., a Gaussian process as surrogate model for all solely numerical problems and a random forest for the problems with mixed and dependent parameter spaces (*hpnnet* and *hpdbnet*). We deviate from the defaults only for the initial design of *hpnnet* and *hpdbnet*. Here, the number of allowed function evaluations compared to the dimension of the parameter set is very small, therefore the initial design has only size $2d$ instead of the default $4d$. Spearmint uses an internal dummy encoding of all categorical parameters for its Gaussian process. The number of iterations on each benchmark as well as the specific settings of all other optimizers are defined in HPOlib.

Evaluation. The results of ten runs on each benchmark are summarized in Figure 5. On each of the four synthetic test functions, `mlrMBO` outperforms both SMAC and TPE and has similar performance compared to Spearmint, except for *michalewicz* where `mlrMBO` outperforms all competitors. For the grid optimizations, `mlrMBO` also performs exceptionally well on every single one, while each other optimizer results in worse performance on at least one of the three problems, which overall places `mlrMBO` on the first place in numeric settings (cf. Table 2). Regarding the neural network and deep belief network, `mlrMBO` achieves similar results as SMAC and slightly better results than Spearmint

[1]Since `BayesOpt` is neither connected to HPOlib nor possesses an `R` interface, we refer the reader to the benchmarks in [16] and do not consider `BayesOpt` in our analysis.

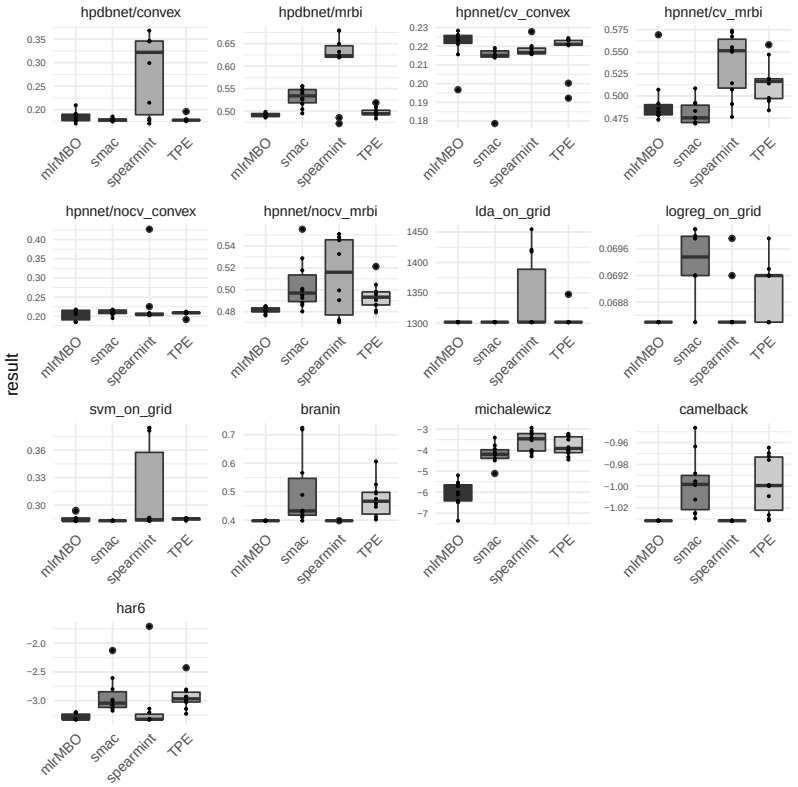

Figure 5: Best objective value (on y axis) found by respective optimizer on 13 HPOlib test functions. For details on the test functions we refer to [8] for *hpnnet* and *hpdbnet* and to [44] for everything else.

and TPE. Especially Spearmint has a clearly worse performance on three of the six problems, while `mlrMBO` is only worse than its competitors on *hpdbnet/cv_convex*. As a result, Table 2 shows that `mlrMBO` also places first w.r.t. aggregated mean ranks for mixed hyperparameter spaces. We can clearly see that `mlrMBO` is on par with other state-of-the-art Bayesian optimization software, even in highly complex settings.

4.3. Model-Based Multi-Objective Optimization

In the following section we will briefly summarize a benchmark performed by Horn et al. [4]. We will focus on the results concerning the four MBMO algorithms implemented in `mlrMBO`. For the detailed experimental setup and further details we refer to the original paper.

Optimizer	Avg. rank	Avg. rank (numeric only)	Avg. rank (mixed only)
`mlrMBO`	1.90 **(1)**	1.64 **(1)**	2.20 **(1)**
`smac`	2.65 **(3)**	2.90 **(3)**	2.35 **(2)**
`spearmint`	2.61 **(2)**	2.32 **(2)**	2.95 **(4)**
`TPE`	2.85 **(4)**	3.14 **(4)**	2.50 **(3)**

Table 2: Average ranks on HPOlib problems, Results were ranked in each replication and then averaged over the replications and problems. Numeric only ranks are based on benchmark 7 to 13 and mixed only ranks are based on 1 to 6.

Benchmarks. The benchmark was performed on nine artificial test functions. On the one hand, four well known multi-objective test functions were used, namely *ZDT1*, *ZDT2* and *ZDT3* each with dimension $d = 5$ and number of objectives $k = 2$, as well as *DTLZ2* using $d = 5$ and $k \in \{2, 5\}$. On the other hand, four additional test functions were constructed by combining various single-objective problems, in such a way that each combination of $d \in \{2, 5\}$ and $k \in \{2, 5\}$ is used. These are called *GOMOP* in the following.

Setup. To simulate an expensive setting, all algorithms had a budget of only $40d$ function evaluations. The popular popular evolutionary multi-objective algorithm NSGA2 [45] and a random search serve as baseline for the four implementation in `mlrMBO`: SMS-EGO, ϵ-EGO, SMS-Ego, and MSPOT (cf. Section 2.6).

Evaluation. Various performance measures for comparing different approximations have been introduced. The most popular measure may be the dominated hypervolume (also known as S-Metric). In the bi-objective case the hypervolume simply measures the area between the discrete approximation of the Pareto front and a pessimistic reference point. If an approximation reaches a higher hypervolume value, it is considered superior.

The final Pareto front approximations were normalized to the interval $[1, 2]^m$ with respect to a reference set. In Figure 6, the hypervolume values of all runs with respect to the reference point $(1.1)^k$ are shown for 20 replications per test function. We see that most MBMO algorithms outperform both baseline algorithms on nearly all test functions, only ParEGO fails occasionally. Moreover, its performance is always worse than ϵ-EGO and SMS-EGO. The three remaining MBMO algorithms perform comparably well, whereas SMS-EGO shows top or near-top performance on all test functions.

5. Conclusion

We introduced the `R` package `mlrMBO`, a modular toolbox for model-based optimization in the `R` programming language. We gave a brief introduction to software specific aspects and features. Furthermore, we performed comprehensive benchmarks of `mlrMBO` against other black-box optimizers in different

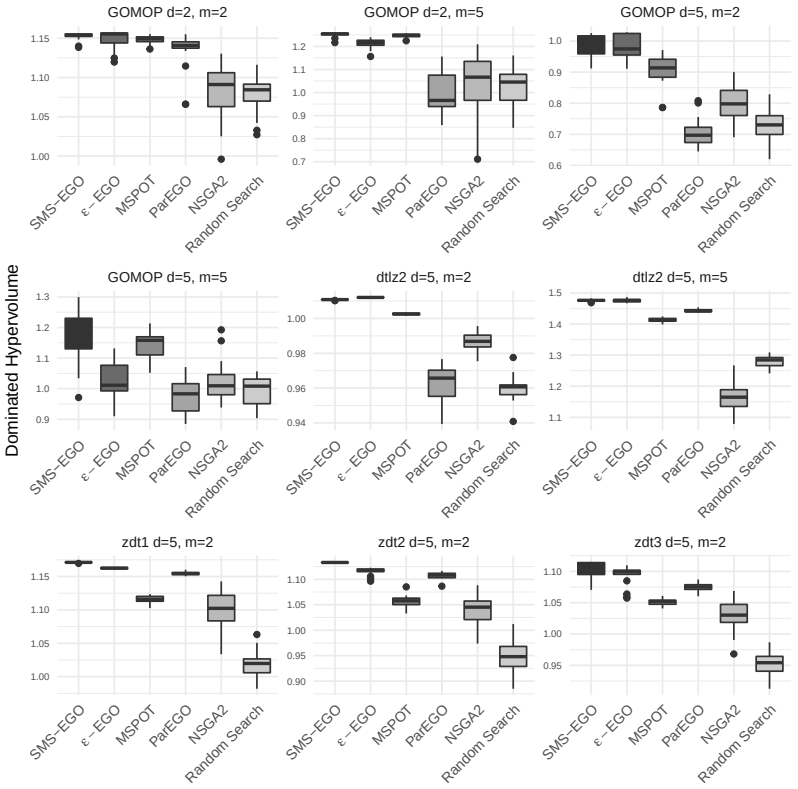

Figure 6: Normalized dominated hypervolume (on y axis) for the respective algorithm on respective test functions.

scenarios. In the single-objective benchmark `mlrMBO` proved state-of-the-art performance regarding solution quality in comparison with the CMA evolutionary strategy, random search, and alternative SMBO implementations, while still being reasonably fast. Furthermore, `mlrMBO` is on par with the well known optimization frameworks SMAC, Spearmint, and TPE as shown by benchmarks using HPOlib. The benchmark study on expensive multi-objective optimization revealed SMBO-based methods, in particular SMS-EGO, to show excellent performance. Both the state-of-the-art NSGA-II evolutionary algorithm as well as the baseline random search algorithm were outperformed on all nine test functions (only ParEGO occasionally failed). All in all the results demonstrate the suitability of the `mlrMBO` toolbox in particular for expensive optimization scenarios in `R` for single- and multi-objective tasks, with continuous or mixed parameter spaces.

Acknowledgments

Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", project A3 (http://sfb876.tu-dortmund.de) and by the Competence Network for Technical, Scientific High Performance Computing in Bavaria (KONWIHR) in the project "Implementierung und Evaluation eines Verfahrens zur automatischen, massiv-parallelen Modellselektion im Maschinellen Lernen".

References

1. Sieben, B., Wagner, T., Biermann, D.. Empirical modeling of hard turning of aisi 6150 steel using design and analysis of computer experiments. *Production Engineering* 2010;4(2):115–125.

2. Sacks, J., Welch, W.J., Mitchell, T.J., Wynn, H.P.. Design and analysis of computer experiments. *Statistical science* 1989;4(4):409–423.

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

4. Horn, D., Wagner, T., Biermann, D., Weihs, C., Bischl, B.. Model-based multi-objective optimization: Taxonomy, multi-point proposal, toolbox and benchmark. In: Gaspar-Cunha, A., Henggeler Antunes, C., Coello, C.C., eds. *Evolutionary Multi-Criterion Optimization*; vol. 9018 of *Lecture Notes in Computer Science*. Springer; 2015:64–78.

5. Ginsbourger, D., Le Riche, R., Carraro, L.. Kriging is well-suited to parallelize optimization. In: *Computational Intelligence in Expensive Optimization Problems*. Springer; 2010:131–162.

6. Bischl, B., Wessing, S., Bauer, N., Friedrichs, K., Weihs, C.. MOI-MBO: Multiobjective infill for parallel model-based optimization. In: *Learning and Intelligent Optimization Conference*. 2014:173–186.

7. Hutter, F., Hoos, H.H., Leyton-Brown, K.. Sequential model-based optimization for general algorithm configuration. In: *LION 5*. 2011:507–523.

8. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.. Algorithms for hyper-parameter optimization. In: *Advances in Neural Information Processing Systems*. 2011:2546–2554.

9. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: *Proceedings of ACM SIGKDD*. 2013:847–855.

10. Lang, M., Kotthaus, H., Marwedel, P., Weihs, C., Rahnenführer, J., Bischl, B.. Automatic model selection for high-dimensional survival analysis. *Journal of Statistical Computation and Simulation* 2015;85(1):62–76.

11. Horn, D., Bischl, B.. Multi-objective parameter configuration of machine learning algorithms using model-based optimization. *Symposium Series on Computational Intelligence* 2016;ACCEPTED.

12. Roustant, O., Ginsbourger, D., Deville, Y.. DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization. *Journal of Statistical Software* 2012;51(1):1–55.

13. Yan, Y.. rBayesianOptimization: Bayesian Optimization of Hyperparameters; 2016. URL: `https://CRAN.R-project.org/package=rBayesianOptimization`; R package version 1.0.0.

14. Snoek, J., Larochelle, H., Adams, R.P.. Practical bayesian optimization of machine learning algorithms. In: *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc.; 2012:2951–2959.

15. Hernández-Lobato, D., Hernández-Lobato, J.M., Shah, A., Adams, R.P.. Predictive entropy search for multi-objective bayesian optimization. In: *Proceedings of the 33nd International Conference on Machine Learning (ICML)*. 2016:1492–1501.

16. Martinez-Cantin, R.. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research* 2014;15:3915–3919.

17. Bartz-Beielstein, T., Zaefferer, M.. A gentle introduction to sequential parameter optimization. Tech. Rep. 2; Bibliothek der Fachhochschule Koeln; 2012. URL: `http://opus.bsz-bw.de/fhk/volltexte/2012/19`.

18. Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M.. mlr: Machine learning in R. *Journal of Machine Learning Research* 2016;17(170):1–5.

19. Koch, P., Bischl, B., Flasch, O., Bartz-Beielstein, T., Weihs, C., Konen, W.. Tuning and evolution of support vector kernels. *Evolutionary Intelligence* 2012;5(3):153–170.

20. Bischl, B., Schiffner, J., Weihs, C.. Benchmarking classification algorithms on high-performance computing clusters. In: Spiliopoulou, M., Schmidt-Thieme, L., Janning, R., eds. *Data Analysis, Machine Learning and Knowledge Discovery*. Studies in Classification, Data Analysis, and Knowledge Organization; Springer; 2014:23–31.

21. Hess, S., Wagner, T., Bischl, B.. PROGRESS: Progressive reinforcement-learning-based surrogate selection. In: Nicosia, G., Pardalos, P., eds. *Learning and Intelligent Optimization*. Lecture Notes in Computer Science; Springer; 2013:110–124.

22. Horn, D., Demircioğlu, A., Bischl, B., Glasmachers, T., Weihs, C.. A comparative study on large scale kernelized support vector machines. *Advances in Data Analysis and Classification* 2016;:1–17.

23. Steponavičė, I., Shirazi-Manesh, M., Hyndman, R.J., Smith-Miles, K., Villanova, L.. On Sampling Methods for Costly Multi-Objective Black-Box Optimization. In: Pardalos, P.M., Zhigljavsky, A., Žilinskas, J., eds. *Advances in Stochastic and Deterministic Global Optimization*. No. 107 in Springer Optimization and Its Applications; Springer International Publishing. ISBN 978-3-319-29973-0 978-3-319-29975-4; 2016:273–296.

24. McKay, M.D., Beckman, R.J., Conover, W.J.. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* 2000;42(1):55–61.

25. Schiffner, J., Bischl, B., Lang, M., Richter, J., Jones, Z.M., Probst, P., Pfisterer, F., Gallo, M., Kirchhoff, D., Kühn, T., Thomas, J., Kotthoff, L.. mlr tutorial. 2016. `arXiv:arXiv:1609.06146`.

26. Bossek, J.. cmaesr: Covariance Matrix Adaptation Evolution Strategy; 2016. URL: `https://CRAN.R-project.org/package=cmaesr`; R package version 1.0.1.

27. Rasmussen, C.E., Williams, C.K.I.. Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning; MIT Press; 2006.

28. Vazquez, E., Bect, J.. Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. *Journal of Statistical Planning and Inference* 2010;140(11):3088–3095.

29. Jones, D.R.. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization* 2001;21(4):345–383.

30. Zhou, Q., Qian, P.Z., Zhou, S.. A simple approach to emulation for computer models with qualitative and quantitative factors. *Technometrics* 2012;.

31. Ding, Y., Simonoff, J.S.. An investigation of missing data methods for classification trees applied to binary response data. *Journal of Machine Learning Research* 2010;11:131–170. URL: `http://www.jmlr.org/papers/v11/ding10a.html`.

32. Sexton, J., Laake, P.. Standard errors for bagged and random forest estimators. *Computational Statistics & Data Analysis* 2009;53(3):801–811.

33. Wager, S., Hastie, T., Efron, B.. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. *Journal of Machine Learning Research* 2014;15:1625–1651. URL: `http://jmlr.org/papers/v15/wager14a.html`.

34. Hutter, F., Hoos, H.H., Leyton-Brown, K.. Parallel algorithm configuration. In: *Learning and Intelligent Optimization*. Springer; 2012:55–70.

35. Preuss, M.. Considerations of budget allocation for sequential parameter optimization (spo. In: *Workshop on Empirical Methods for the Analysis of Algorithms, Proceedings*. 2006:35–40.

36. Picheny, V., Ginsbourger, D., Richet, Y., Caplin, G.. Quantile-based optimization of noisy computer experiments with tunable precision. *Technometrics* 2013;55(1):2–13.

37. Picheny, V., Wagner, T., Ginsbourger, D.. A benchmark of kriging-based infill criteria for noisy optimization. *Struct Multidiscip Optim* 2013;48(3):607–626.

38. Knowles, J.. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation* 2006;10(1):50–66.

39. Zaefferer, M., Bartz-Beielstein, T., Naujoks, B., Wagner, T., Emmerich, M.. A case study on multi-criteria optimization of an event detection software under limited budgets. In: Purshouse, R., et al., eds. *Proc. 7th Int'l. Conf. Evolutionary Multi-Criterion Optimization (EMO)*. Springer; 2013:756–770.

40. Ponweiser, W., Wagner, T., Biermann, D., Vincze, M.. Multiobjective optimization on a limited amount of evaluations using model-assisted $\mathcal{S}$-metric selection. In: *Proc. 10th Int'l Conf. Parallel Problem Solving from Nature (PPSN)*. 2008:784–794.

41. Wagner, T.. Planning and Multi-Objective Optimization of Manufacturing Processes by Means of Empirical Surrogate Models. Vulkan Verlag, Essen; 2013.

42. Bossek, J.. smoof: Single and Multi-Objective Optimization Test Functions; 2016. URL: `https://github.com/jakobbossek/smoof`; R package version 1.4.

43. Bischl, B., Lang, M., Mersmann, O., Rahnenführer, J., Weihs, C.. BatchJobs and BatchExperiments: Abstraction mechanisms for using R in batch environments. *Journal of Statistical Software* 2015;64(11):1–25.

44. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In: *NIPS workshop on Bayesian Optimization in Theory and Practice*. 2013:1–5.

45. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 2002;6(2):182–197.

6. RAMBO: Resource-Aware Model-Based Optimization with Scheduling for Heterogeneous Runtimes and a comparison with Asynchronous Model-Based Optimization

Declaration of the specific contributions of the author The RAMBO method was proposed by Helena Kotthaus and Jakob Richter. They wrote the major part of the code, conducted the benchmarks and wrote the corresponding sections of the publication. The author mainly contributed to the related work section of the paper with the general overview of algorithms for synchronous and asynchronous parallel Bayesian optimization, as well as general discussions on the structure of the benchmarks. Bernd Bischl and Michel Lang also helped with the implementation and software design in `mlrMBO`. Bernd Bischl, Michel Lang, Jörg Rahneführer and Peter Marwedel helped to improve text, motivation for the proposed approach as well as the scientific context.

RAMBO: Resource-Aware Model-Based Optimization with Scheduling for Heterogeneous Runtimes and a Comparison with Asynchronous Model-Based Optimization

Helena Kotthaus[1(✉)], Jakob Richter[2], Andreas Lang[1], Janek Thomas[3], Bernd Bischl[3], Peter Marwedel[1], Jörg Rahnenführer[2], and Michel Lang[2]

[1] Department of Computer Science 12, TU Dortmund University, Dortmund, Germany
helena.kotthaus@tu-dortmund.de
[2] Department of Statistics, TU Dortmund University, Dortmund, Germany
[3] Department of Statistics, LMU München, Munich, Germany

Abstract. Sequential model-based optimization is a popular technique for global optimization of expensive black-box functions. It uses a regression model to approximate the objective function and iteratively proposes new interesting points. Deviating from the original formulation, it is often indispensable to apply parallelization to speed up the computation. This is usually achieved by evaluating as many points per iteration as there are workers available. However, if runtimes of the objective function are heterogeneous, resources might be wasted by idle workers. Our new knapsack-based scheduling approach aims at increasing the effectiveness of parallel optimization by efficient resource utilization. Derived from an extra regression model we use runtime predictions of point evaluations to efficiently map evaluations to workers and reduce idling. We compare our approach to five established parallelization strategies on a set of continuous functions with heterogeneous runtimes. Our benchmark covers comparisons of synchronous and asynchronous model-based approaches and investigates the scalability.

Keywords: Black-box optimization · Model-based optimization · Global optimization · Resource-aware scheduling · Performance management · Parallelization

1 Introduction

Efficient global optimization of expensive black-box functions is of interest to many fields of research. In the engineering industry, computationally expensive models have to be optimized; for machine learning hyperparameters have to be tuned; and for computer experiments in general, expensive algorithms have parameters that have to be optimized to obtain a well-performing algorithm configuration. The problems of global optimization can usually be modeled by a

R. Battiti et al. (Eds.): LION 2017, LNCS 10556, pp. 180–195, 2017.
https://doi.org/10.1007/978-3-319-69404-7_13

real-valued objective function f with a d-dimensional domain space. The challenge is to find the best point possible within a very limited time budget.

Together with [1,22,23], Model-based optimization (MBO) [15] is a state-of-the-art algorithm for *expensive* black-box functions. Starting on an initial design of already evaluated configurations, a regression model guides the search to new configurations by predicting the outcome of the black-box on yet unseen configurations. Based on this prediction an infill criterion (also called acquisition function) proposes a new promising configuration for evaluation. In each iteration the regression model is updated on the evaluated configurations of the previous iteration until the budget is exhausted. Jones et al. [15] proposed this now popular *Efficient Global Optimization* (EGO) algorithm. EGO sequentially adds points to the design using Kriging as a surrogate and the *Expected Improvement* (EI) as an infill criterion. Following, other infill criteria [14], specializations e.g. for categorical search spaces like in SMAC [10] and noisy optimization [20] have been introduced.

For computer experiments, parallelization has become of increasing interest to reduce the overall computation time. Originally, the EGO algorithm iteratively proposes one point to be evaluated after another. To allow for parallelization, infill criteria and techniques (constant liar, Kriging believer, qEI [9], qLCB [11], MOI-MBO [4]) have been suggested that propose multiple points in each iteration. Usually, the number of proposed points equals the number of available CPUs. However, these methods still do use the available resources inefficiently if the runtime of the black-box is heterogeneous. Before new proposals can be generated, the results of all evaluations within one iteration are gathered to update the model. Consequently all CPUs have to wait for the slowest function evaluation before receiving a new point proposal. This can lead to idling CPUs that are not contributing to the optimization. The goal in general is to use all available CPU time to solve the optimization problem.

One approach to avoid idling is to desynchronize the model update. Here, the model is updated each time an evaluation has finished, letting each parallel worker propose the next point for evaluation itself. Such asynchronous techniques have been suggested and discussed by [8,12]. The main challenge is to modify the infill criterion to deal with points that are currently under evaluation to avoid evaluations of very similar configurations. The *Expected Expected Improvement* (EEI) [13] is one possibility for such a modification.

Another strategy is to keep the synchronous model update and schedule the evaluations of the proposed points in such a way that idling is reduced. Such an approach was presented in [19]. Here, a second regression model is used to predict runtimes for the proposed points which are used as an input for scheduling.

Our article contains the following contributions: First, we extended the parallel, resource-aware synchronous model-based optimization strategy proposed in [19] with an improved resource-aware scheduling algorithm. This algorithm, which replaces the original simple first fit heuristic, is based on a knapsack solver to better handle heterogeneous runtimes. Furthermore we use a clustering-based refinement strategy to ensure improved spatial diversity of the evaluated points.

Second, we compare our algorithm to three asynchronous MBO strategies that also aim at using all available CPU time to solve the optimization problem in parallel. Two of them [8,12] use Kriging as a surrogate and the third is included in SMAC [10] which uses a random forest surrogate.

Third, we benchmark the MBO algorithms on a set of established continuous test functions combined with simulated runtimes. For each function we use a 2- and a 5-dimensional version each of which is optimized using 4 and 16 CPUs in parallel to investigate scalability.

Compared to the considered asynchronous approaches, our new approach converges faster to the optima if the runtime estimates used as input for scheduling are reliable.

2 Model-Based Global Optimization

The aim of global optimization is to find the global minimum of a given function $f : \mathcal{X} \to \mathbb{R}$, $f(\boldsymbol{x}) = y, \boldsymbol{x} = (x_1, \ldots, x_d)^T$. Here, we assume $\mathcal{X} \subset \mathbb{R}^d$, usually expressed by simple box constraints. The optimization is guided by a surrogate model which estimates the response surface of the black-box function f (see also [22,23]). The surrogate is comparably inexpensive to evaluate and is utilized to propose new promising points $\boldsymbol{x}^*$ in an iterative fashion. A promising point $\boldsymbol{x}^*$ is determined by optimizing some infill criterion. After, $f(\boldsymbol{x}^*)$ is evaluated to obtain the corresponding objective value y, the surrogate model is refitted and a new point is proposed. The infill criterion quantifies the potential improvement based, on an exploitation-exploration trade-off where a low (good) expected value of the solution $\hat{\mu}(\boldsymbol{x})$ is rewarded, and low estimated uncertainty $\hat{s}(\boldsymbol{x})$ is penalized. A popular infill criterion, especially for Kriging surrogate models, is the expected improvement

$$\begin{aligned} \mathrm{EI}(\boldsymbol{x}) &= \mathbb{E}(\max(y_{\min} - \hat{\mu}(\boldsymbol{x}), 0)) \\ &= (y_{\min} - \hat{\mu}(\boldsymbol{x}))\, \Phi\left(\frac{y_{\min} - \hat{\mu}(\boldsymbol{x}))}{\hat{s}(\boldsymbol{x})}\right) + \hat{s}(\boldsymbol{x})\phi\left(\frac{y_{\min} - \hat{\mu}(\boldsymbol{x})}{\hat{s}(\boldsymbol{x})}\right), \end{aligned}$$

where Φ is the distribution and ϕ is the density function of the standard normal distribution and $y_{\min}$ is the best observed function value so far. Alternatively, the comparably simpler lower confidence bound criterion

$$\mathrm{LCB}(\boldsymbol{x}, \lambda) = \hat{\mu}(\boldsymbol{x}) - \lambda \hat{s}(\boldsymbol{x}), \quad \lambda \in \mathbb{R}$$

is used, where $\hat{\mu}(\boldsymbol{x})$ denotes the posterior mean and $\hat{s}(\boldsymbol{x})$ the posterior standard deviation of the regression model at point $\boldsymbol{x}$. Before entering the iterative process, initially some points have to be pre-evaluated. These points are generally chosen in a space-filling manner to uniformly cover the input space. The optimization usually stops after a target objective value is reached or a predefined budget is exhausted [22,23].

2.1 Parallel MBO

Ordinary MBO is sequential by design. However, applications like hyperparameter optimization for machine learning or computer simulations have driven the rapid development of extensions for parallel execution of multiple point evaluations. The parallel extensions either focus on a synchronous model update using infill criteria with multi-point proposals or implement an asynchronous evaluation where each worker generates one new point proposal individually.

Multi-Point proposals derive not only one single point $\boldsymbol{x}^*$ from a surrogate model, but q points $\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_q^*$ simultaneously. The q proposed points must be sufficiently different from each other to avoid multiple evaluations with the same configuration. For this reason Hutter et al. [11] introduced the criterion

$$\text{qLCB}(\boldsymbol{x}, \lambda_j) = \hat{\mu}(\boldsymbol{x}) - \lambda_j \hat{s}(\boldsymbol{x}) \text{ with } \lambda_j \sim \text{Exp}(\lambda) \tag{1}$$

as an intuitive extension of the LCB criterion using an exponentially distributed random variable. Since λ guides the trade-off between exploration and exploitation, sampling multiple different λ_j might result in different "best" points by varying the impact of the standard deviation. The qLCB criterion was implemented in a distributed version of SMAC [11]. An extension of the EI criterion is the qEI criterion [9] which directly optimizes the expected improvement over q points. A closed form solution to calculate qEI exists for $q = 2$ and useful approximations can be applied for $q \leq 10$ [7]. However, as the computation is using Monte Carlo sampling, it is quite expensive. A less expensive and popular alternative is *Kriging believer* approach [9]. Here, the first point is proposed using the single-point EI criterion. Its posterior mean value is treated as a real value of f to refit the surrogate, effectively penalizing the surrounding region with a lower standard deviation for the next point proposal using EI again. This is repeated until q proposals are generated.

In combination with parallel synchronous execution the above described multi-point infill approaches can lead to underutilized systems because a new batch of points can only be proposed as soon as the slowest function evaluation is terminated. Snoek et al. [21] introduce the EI per second to address heterogeneous runtimes. The runtime of a configuration is estimated using a second surrogate model and a combined infill criterion can be constructed which favors less expensive configurations.

We also use surrogate models to estimate resource requirements but instead of adapting the infill criterion, we use them for the scheduling of parallel function evaluations. Our goal is to guide MBO to interesting regions in a faster and resource-aware way without directly favoring less expensive configurations.

Asynchronous Execution approaches the problem of parallelizing MBO from a different angle. Instead of evaluating multiple points in batches to synchronously refit the model, the model is refitted after each function evaluation to increase CPU utilization workers. Here, each worker propose the next point for evaluation itself, even when configurations $\boldsymbol{x}_{\text{busy}}$ are currently under evaluation on other

processing units. The busy evaluations have to be taken into account by the surrogate model to avoid that new point proposals are identical or very similar to pending evaluations. Here, the Kriging believer approach [9] can be applied to block these regions. Another theoretically well-founded way to impute pending values is the *expected* EI (EEI) [8,13,21]. The unknown value of $f(\boldsymbol{x}_{\text{busy}})$ is integrated out by calculating the expected value of y_{busy} via Monte Carlo sampling, which is, similar to qEI, computationally demanding. For each Monte Carlo iteration values $y_{1,\text{busy}}, \ldots, y_{\mu,\text{busy}}$ are drawn from the posterior distribution of the surrogate regression model at $\boldsymbol{x}_{1,\text{busy}}, \ldots, \boldsymbol{x}_{\mu,\text{busy}}$, with μ denoting the number of pending evaluations. These values are combined with the set of already known evaluations and used to fit the surrogate model. The EEI can then simply be calculated by averaging the individual expected improvement values that are formed by each Monte Carlo sample:

$$\widehat{\text{EEI}(\boldsymbol{x})} = \frac{1}{n_{\text{sim}}} \sum_{i=1}^{n_{\text{sim}}} \text{EI}_i(\boldsymbol{x}) \tag{2}$$

whereas n_{sim} denotes the number of Monte Carlo iterations.

Besides the advantage of an increased CPU utilization, asynchronous execution can also potentially cause additional runtime overhead due to the higher number of model refits, especially when the number of workers increases. Therefore our experiments include a comparison with most of the above described approaches to investigate the advantages and disadvantages.

Instead of using asynchronous execution to efficiently utilize parallel computer architectures, our new approach uses the synchronous execution combined with resource-aware scheduling and is presented in the following section.

3 Resource-Aware Scheduling with Synchronous Model Update

The goal of our new scheduling strategy is to guide MBO to interesting regions in a faster and resource-aware way. To efficiently map jobs (proposed points) to available resources our strategy needs to know the resource demands of jobs before execution. Therefore, we estimate the runtime of each job using a regression model. Additionally, we calculate an execution priority for each job based on the multi-point infill criterion. In the following, we will describe these inputs.

3.1 Infill Criterion - Priority

The priorities of the proposed points should reflect their usefulness for optimization. In our setup we opt for the qLCB (1) to generate a set of job proposals by optimizing the LCB for q randomly drawn values of $\lambda_j \sim \text{Exp}(\frac{1}{2})$, as in Richter et al. [19]. qLCB is suitable because the proposals are independent of each other. There is no direct order of the set of obtained candidates $\boldsymbol{x}_j^*$ in terms of how promising or important one candidate is in comparison to each

other. Therefore, we introduce an order that steers the search more towards promising areas. We give the highest priority to the point $\boldsymbol{x}_j$ that was proposed using the smallest value of λ_j. We define the priority for each point as $p_j := -\lambda_j$.

3.2 Resource Estimation

To estimate the resource demands of proposed candidates, we use a separate regression model. To adapt to the domain space of the objective function, we choose the same regression method used for the surrogate. In the same fashion as for the MBO algorithm, runtimes are predicted in each MBO iteration based on all previously evaluated jobs and measured runtimes.

3.3 Resource-Aware Knapsack Scheduling

The goal of our scheduling strategy is to reduce the CPU idle time on the workers while acquiring the feedback of the workers in the shortest possible time to avoid model update delay. The set of points proposed by the multi-point infill criterion forms the set of jobs $J = \{1, \ldots, q\}$ that we want to execute on the available CPUs $K = \{1, \ldots, m\}$. For each job the estimated runtime is given by $\hat{t}_j$ and the corresponding priority is given by p_j. To reduce idle times caused by evaluations of jobs with a low priority, the maximal runtime for each MBO iteration is defined by the runtime of the job with the highest priority. Lower prioritized jobs have to subordinate. At the same time we want to maximize the profit, given by the priorities, of parallel job executions for each model update. To solve this problem, we apply the $0-1$ multiple knapsack algorithm by interfacing the R-package `adagio` for global optimization routines [5]. Here the knapsacks are the available CPUs and their capacity is the maximally allowed computing time, defined by the runtime of the job with the highest priority. The items are the jobs J, their weights are the estimated runtimes $\hat{t}_j$ and their values are the priorities p_j. The capacity for each CPU is accordingly $\hat{t}_{j^*}$, with $j^* := \arg\max_j p_j$. To select the best subset of jobs the algorithm maximizes the profit Q:

$$Q = \sum_{j \in J} \sum_{k \in K} p_j c_{kj},$$

which is the sum of priorities of the selected jobs, under the restriction of the capacity

$$\hat{t}_{j^*} \geq \sum_{j \in J} \hat{t}_j c_{kj} \, \forall k \in K$$

per CPU. The restriction with the decision variable $c_{kj} \in \{0, 1\}$

$$1 \geq \sum_{k \in K} c_{kj} \, \forall j \in J, c_{kj} \in \{0, 1\}.$$

ensures that a job j is at most mapped to one CPU.

As the job with the highest priority defines the time bound $\hat{t}_{j^*}$ it is mapped to the first CPU $k = 1$ exclusively and single jobs with higher runtimes are directly discarded. Then the knapsack algorithm is applied to assign the remaining candidates in J to the remaining $m-1$ CPUs. This leads to the best subset of J that can be run in parallel minimizing the delay of the model update. If a CPU is left without a job we query the surrogate model for a job with an estimated runtime smaller or equal to $\hat{t}_{j^*}$ to fill the gaps. For a useful scheduling the set of candidates should have considerably more candidates q than available CPUs. This knapsack scheduling is a direct enhancement of the first fit scheduling strategy presented in [19].

3.4 Refinement of Job Priorities via Clustering

The refinement of job priorities has the goal to avoid parallel evaluations of very similar configurations. Approaches to specifically propose points that are promising but yet diverse are described in [4]. qLCB performed well and was chosen here because it is comparably inexpensive to create many independent candidates. However, qLCB does not include a penalty for the proximity of selected points which gets problematic if the number of parallel points is high. Therefore, we use a distance measure to reprioritize p_j to $\tilde{p}_j$, encouraging the selection sets of candidates more scattered in the domain space.

First, we oversample a set of $q > m$ candidate points from the qLCB criterion and partition them into $\tilde{q} < q$ clusters using the Euclidean distance. Next, we take the candidate with maximum priority p_j from each cluster and sort them according to their priority before pushing them to the list $\tilde{J}$ of selected jobs. Selected jobs are removed from the clusters and empty clusters are eliminated. We repeat this procedure until we have moved all q jobs into the list $\tilde{J}$. Finally, we assign new priorities $\tilde{p_j}$ based on the order of $\tilde{J}$, i.e. the first job in $\tilde{J}$ gets the highest priority q and the last job gets the lowest priority 1.

As a result, the set of candidates contains batches of jobs with similar priority that are spread in the domain space. The new priorities serve as input for scheduling which groups the q jobs to m CPUs using the runtime estimates $\hat{t}$.

4 Numerical Experiments

In our experiments, we consider two categories of synthetic functions to ensure a fair comparison in a disturbance-free environment. They are implemented in the `R` package `smoof` [6]:

1. Functions with a smooth surface: `rosenbrock`(d) and `bohachevsky`(d) with dimension $d = 2, 5$. They are likely to be fitted well by the surrogate.
2. Highly multimodal functions: `ackley`(d) and `rastrigin`(d) $(d = 2, 5)$. We expect that surrogate models can have problems to achieve a good fit here.

As these are illustrative test functions, they have no significant runtime. As a resort, we also use these functions to simulate runtime behavior. First, we

combine two functions: One determines the number of seconds it would take to calculate the objective value of the other function. E.g., for the combination `rastrigin`(2)`.rosenbrock`(2) it would require `rosenbrock`(2)($\boldsymbol{x}$) seconds to retrieve the objective value `rastrigin`(2)($\boldsymbol{x}$) for an arbitrary proposed point $\boldsymbol{x}$. Technically, we just sleep `rosenbrock`(2)($\boldsymbol{x}$) seconds before returning the objective. We simulate the runtime with either `rosenbrock`(d) or `rastrigin`(d) and analyze all combinations of our four objective functions, except where the objective and the time function are identical.

A prerequisite for this approach is the unification of the input space. Thus, we simply mapped values from the input space of the objective function to the input space of the time function. The output of the time functions is scaled to return values between 5 min to 60 min.

We examine the capability of the considered optimization strategies to minimize functions with highly heterogeneous runtimes within a limited time budget. To do this, we measure the distance between the best found point at time t and a predefined target value. We call this measure accuracy. In order to make this measure comparable across different objective functions, we scale the function values to $[0, 1]$ with zero being the target value. It is defined as the best y reached by any optimization method after the complete time budget. The upper bound 1 is the best y found in the initial design (excluding the initial runs of `smac`) which is identical for all algorithms per given problem. Both values are averaged over the 10 replications.

If an algorithm needs 2 h to reach an accuracy of 0.5, this means that within 2 h half of the way to 0 has been accomplished, after starting at 1. We compare the differences between optimizers at the three accuracy levels 0.5, 0.1 and 0.01.

The optimizations are repeated 10 times and conducted on $m = 4$ and $m = 16$ CPUs. We allow each optimization to run for 4 h on 4 CPUs and for 2 h on 16 CPUs in total which includes all computational overhead and idling. All computations were performed on a Docker Swarm cluster using the `R` package `batchtools` [18]. The initial design is generated by Latin hypercube sampling with $n = 4 \cdot d$ points and all of the following optimizers start with the same design in the respective repetition:

- `rs`: Random search, serving as base-line.
- `qLCB`: Synchronous approach using qLCB where in each iteration $q = m$ points are proposed.
- `ei.bel`: Synchronous approach using Kriging believer where in each iteration m points are proposed.
- `asyn.eei`: Asynchronous approach using EEI (100 Monte Carlo iterations)
- `asyn.ei.bel`: Asynchronous Kriging believer approach.
- `rambo`: Synchronous approach using qLCB with our new scheduling approach where in each iteration $q = 8 \cdot m$ candidates are proposed.

`qLCB` and `ei.bel` are implemented in the `R` package `mlrMBO` [3], which builds upon the machine learning framework `mlr` [2]. `asyn.eei`, `asyn.ei.bel` and `rambo` are also based on `mlrMBO`. We use a Kriging model from the package `DiceKriging` [20] with a Matern$\frac{5}{2}$-kernel for all approaches above and add a

nugget effect of $10^{-8} \cdot \mathrm{Var}(\boldsymbol{y})$, where $\boldsymbol{y}$ denotes the vector of all observed function outcomes. Additionally we compare our implementations to:

`smac`: Asynchronous approach that turns m independent SMAC runs into m dependent runs by sharing surrogate model data (also called shared-model-mode[1]).

SMAC was allowed the same initial budget as the other optimizers and was started with the defaults and the shared-model-mode activated. SMAC uses a random forest as surrogate and the EI criterion.

4.1 Quality of Resource Estimation

The quality of resource-aware scheduling naturally depends on the accuracy of the resource estimation. Without reliable runtime predictions, the scheduler is unable to optimize for efficient utilization. As Fig. 1 exemplary shows, the runtime prediction for the `rosenbrock`(5) time function works well as the residual values are getting smaller over time, while the runtime prediction for `rastrigin`(5) is comparably imprecise. For the 2-dimensional versions the results are similar. This encourages to consider scenarios separately where runtime prediction is possible (`rosenbrock`(·), Subsect. 4.2) and settings where runtime prediction is error-prone (`rastrigin`(·), Subsect. 4.3) for further analysis.

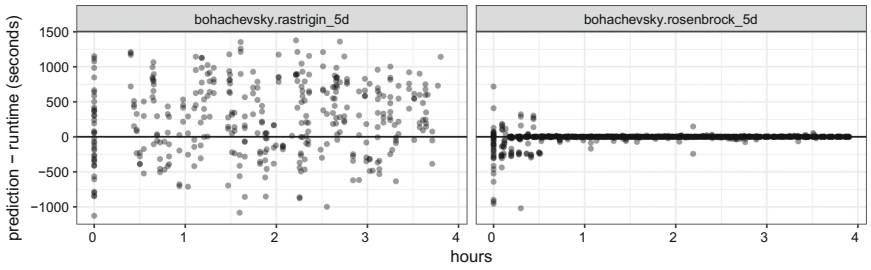

Fig. 1. Residuals of the runtime prediction in the course of time for the `rosenbrock`(5) and `rastrigin`(5) time functions on 4 CPUs and `bohachevsky`(5) as objective function. Positive values indicate an overestimated runtime and negative values an underestimation.

4.2 High Runtime Estimation Quality: `rosenbrock`

Figure 2 shows boxplots for the time required to reach the three different accuracy levels in 10 repetitions within a budget of 4 h real time on 4 CPUs (upper part) and

[1] Hutter, F., Ramage, S.: Manual for SMAC version v2.10.03-master. Department of Computer Science, UBC. (2015), www.cs.ubc.ca/labs/beta/Projects/SMAC/v2.10.03/manual.pdf.

Table 1. Ranking for accuracy levels 0.5, 0.1, 0.01 averaged over all problems with `rosenbrock`(·) time function on 4 and 16 CPUs with a time budget of 4 h and 2 h, respectively.

Algorithm	4 CPUs			16 CPUs		
	0.5	0.1	0.01	0.5	0.1	0.01
`asyn.eei`	3.32 (2)	3.52 (1)	4.97 (2)	3.75 (3)	4.30 (3)	5.45 (3)
`asyn.ei.bel`	3.55 (3)	4.10 (3)	4.97 (2)	3.48 (2)	4.08 (2)	4.53 (2)
`RAMBO`	3.17 (1)	3.85 (2)	4.57 (1)	3.13 (1)	3.93 (1)	4.47 (1)
`ei.bel`	4.38 (4)	4.98 (4)	5.90 (5)	5.00 (5)	5.48 (6)	6.28 (6)
`qLCB`	4.52 (5)	5.03 (5)	5.63 (4)	4.72 (4)	5.17 (4)	6.10 (4)
`rs`	6.02 (6)	6.67 (6)	6.83 (7)	5.50 (7)	6.48 (7)	6.87 (7)
`smac`	6.22 (7)	6.70 (7)	6.82 (6)	5.32 (6)	5.47 (5)	6.17 (5)

2 h on 16 CPUs (lower part). The faster an optimizer reaches the desired accuracy level, the lower the displayed box and the better the approach. If an algorithm did not reach an accuracy level within the time budget, we impute with the respective time budget (4 h or 2 h) plus a penalty of 1000 s.

Table 1 lists the aggregated ranks over all objective functions, grouped by algorithm, accuracy level, and number of CPUs. For this computation, the algorithms are ranked w.r.t. their performance for each replication and problem before they are aggregated with the mean. If there are ties (e.g. if an accuracy level was not reached), all values obtain the worst possible rank.

The benchmarks indicate an overall advantage of our proposed resource-aware MBO algorithm (`rambo`): On average, `rambo` reaches the accuracy level first in 2 of 3 setups on 4 CPUs and is always fastest on 16 CPUs. `rambo` is closely followed by the asynchronous variant `asyn.eei` on 4 CPUs but the lead becomes more clear on 16 CPUs. In comparison to the conventional synchronous algorithms (`ei.bel`, `qLCB`), `rambo` as well as `asyn.eei` and `asyn.ei.bel` reach the given accuracy levels in shorter time. This is especially true for objective functions that are hard to model (`ackley`(·), `rastrigin`(·)) by the surrogate as seen in Fig. 2. The simpler `asyn.ei.bel` performs better than `asyn.eei` on 16 CPUs. Except for `smac`, all presented MBO methods outperform base-line `rs` on almost all problems and accuracy levels. The bad average results for `smac` are partly due to its low performance on the $5d$ problems and probably because of the disadvantage of using a random forest as a surrogate on purely numerical problems. A recent benchmark in [3] was able to demonstrate the competitive performance of the Kriging based EGO approach. On 16 CPUs `smac` performs better than `rs` and comparable to `ei.bel`.

For a thorough analysis of the optimization, Fig. 3 exemplary visualizes the mapping of the parallel point evaluations (jobs) for all MBO algorithms on 16 CPUs for the $5d$ versions of the problems. Each gray box represents computation of a job on the respective CPU. For the synchronous MBO algorithms

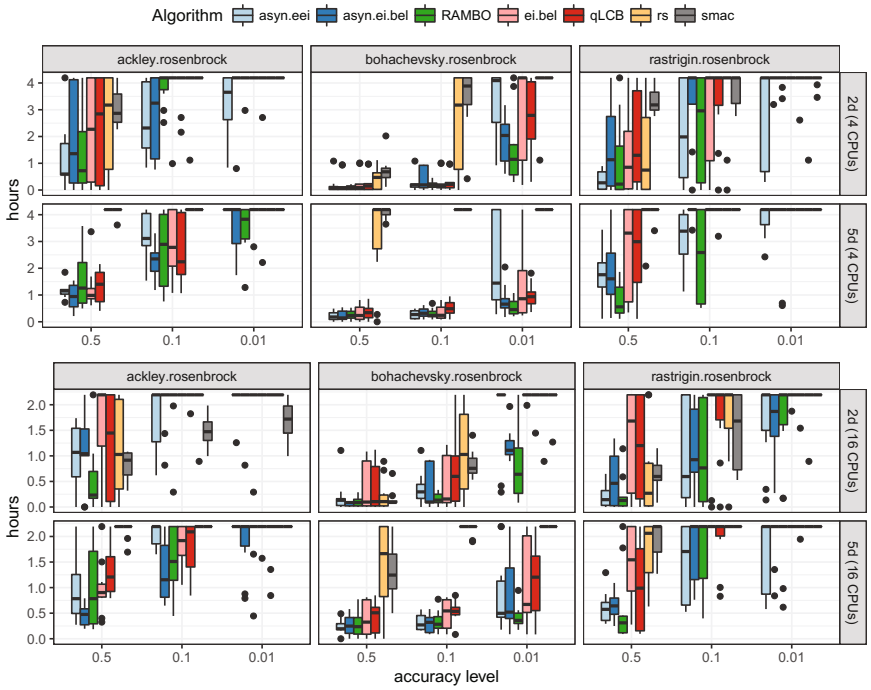

Fig. 2. Accuracy level vs. execution time for different objective functions using time function `rosenbrock`(·) (lower is better).

(`rambo`, `qLCB`, `ei.bel`) the vertical lines indicate the end of an MBO iteration. For `asyn.eei` red boxes indicate that the CPU is occupied with the point proposal. The necessity of a resource estimation for jobs with heterogeneous runtimes becomes obvious, as `qLCB` and `ei.bel` can cause long idle times by queuing jobs together with large runtime differences. The knapsack scheduling in `rambo` manages to clearly reduce this idle time. This effect of efficient resource utilization increases with the number of CPUs. `rambo` reaches nearly the same effective resource-utilization as the asynchronous `asyn.ei.bel` algorithm and `smac` (see Fig. 3) and at the same time reaches the accuracy level fastest on 16 CPUs.

The Monte Carlo approach `asyn.eei` generates a high computational overhead as indicated by the red boxes, which reduces the effective number of evaluations. Idling occurs because the calculation of the EEI is encouraged to wait for ongoing EEI calculations to include their proposals. This overhead additionally increases with the number of already evaluated points. `asyn.ei.bel` and `smac` have a comparably low overhead and thus basically no idle time. This seems to be an advantage for `asyn.ei.bel` on 16 CPUs where it performs better on average than its complex counterpart `asyn.eei`.

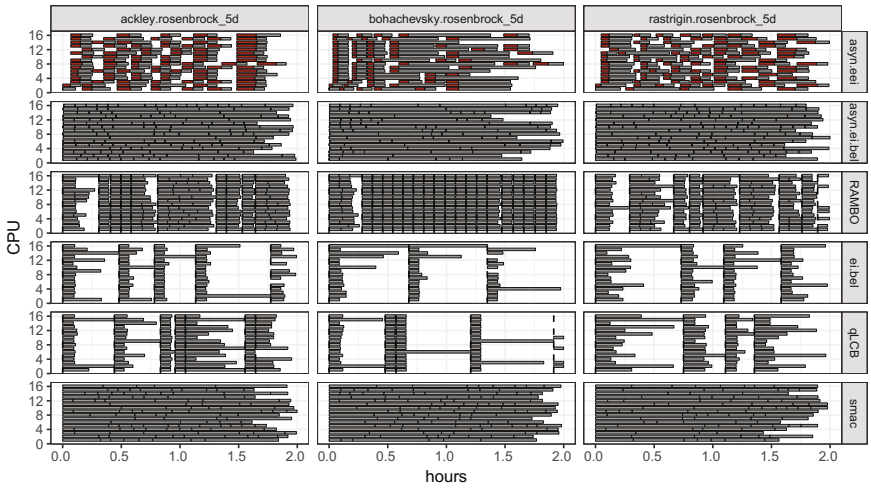

Fig. 3. Scheduling of MBO algorithms. Time on x-axis and mapping of candidates to $m = 16$ CPUs on y-axis. Each gray box represents a job. Each red box represents overhead for the asynchronous approaches. The gaps represent CPU idle time. (Color figure online)

Summed up, if the resource estimation that is used in `rambo` has a high quality, `rambo` clearly outperforms the considered synchronous MBO algorithms `qLCB`, `ei.bel`, and `smac`. This indicates, that the resource utilization obtained by the scheduling in `rambo` leads to faster and better results, especially, when the number of available CPUs increases. On average `rambo` performs better than all considered asynchronous approaches.

4.3 Low Runtime Estimation Quality: `rastrigin`

The time function `rastrigin` used in the following scenario is difficult to fit by surrogate models, as visualized by the residual plot in Fig. 1. For this reason, the benefit of our resource-aware knapsack strategy is expected to be minimal. For example in a possible worst case multiple supposedly short jobs are assigned to one CPU but their real runtime is considerably longer.

Similar to Subsect. 4.2, Fig. 4 shows boxplots for the benchmark results, but with `rastrigin`$(\cdot)$ as the time function. Table 2 provides the mean ranks for Fig. 4, calculated in the same way as in previous Subsect. 4.2.

Despite possible wrong scheduling decisions, `rambo` still manages to outperform `qLCB` and performs better than `ei.bel` on the highest accuracy level on average. `asyn.eei` reaches the accuracy levels fastest on 4 CPUs. Similar to the previous benchmarks on Subsect. 4.2, the simplified `asyn.ei.bel` seems to benefit from its reduced overhead and places first on 16 CPUs. This difference w.r.t. the scalability becomes especially visible on `rosenbrock`$(\cdot)$.

Table 2. Ranking for accuracy levels 0.5, 0.1, 0.01 averaged over all problems with `rastrigin`(·) time function on 4 and 16 CPUs with a time budget of 4 h and 2 h, respectively.

Algorithm	4 CPUs			16 CPUs		
	0.5	0.1	0.01	0.5	0.1	0.01
`asyn.eei`	3.65 (1)	3.25 (1)	4.47 (1)	4.42 (3)	4.38 (2)	5.20 (3)
`asyn.ei.bel`	3.88 (2)	3.50 (2)	4.52 (2)	3.90 (1)	3.75 (1)	4.33 (1)
`RAMBO`	4.50 (4)	4.70 (4)	4.72 (3)	4.43 (4)	4.60 (4)	5.17 (2)
`ei.bel`	4.22 (3)	4.42 (3)	4.87 (4)	4.33 (2)	4.55 (3)	5.27 (4)
`qLCB`	4.95 (5)	4.80 (5)	5.38 (5)	5.10 (5)	5.00 (5)	5.82 (5)
`rs`	6.30 (7)	6.42 (6)	6.63 (6)	5.78 (7)	6.23 (7)	6.43 (6)
`smac`	5.90 (6)	6.98 (7)	7.00 (7)	5.30 (6)	5.77 (6)	6.72 (7)

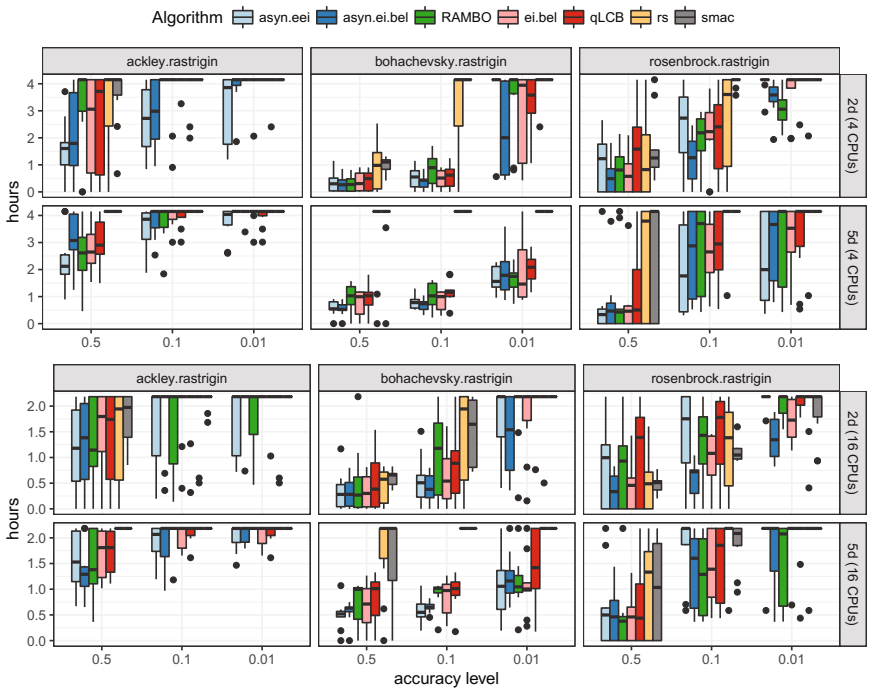

Fig. 4. Accuracy level vs. execution time for different objective functions using time function `rastrigin`(·) (lower is better).

`smac` can not compete with the Kriging based optimizers. Overall, `rambo` appears not to be able to outperform the asynchronous MBO methods on 4 CPUs as unreliable runtime estimates likely lead to suboptimal scheduling decisions.

However, `rambo` reaches comparable results to `asyn.eei` on 16 CPUs and compared to the default synchronous approaches it is a viable choice.

5 Conclusion

We benchmarked our knapsack based resource-aware parallel MBO algorithm `rambo` against popular synchronous and asynchronous MBO approaches on a set of illustrative test functions for global optimization methods. Our new approach was able to outperform SMAC and the default synchronous MBO approach `qLCB` on the continuous benchmark functions. On setups with high runtime estimation quality it converged faster to the optima than the competing MBO algorithms on average. This indicates, that the resource utilization obtained by our new approach improves MBO, especially, when the number of available CPUs increases. On setups with low runtime estimation quality the asynchronous Kriging based approaches performed best on 4 CPUs and only the simplified asynchronous Kriging believer kept its lead on 16 CPUs. Unreliable estimates likely lead to suboptimal scheduling decisions for `rambo`. While the asynchronous Kriging believer approach, SMAC and `rambo` benefited from increasing the number of CPUs, the overhead of the asynchronous approach based on EEI increased.

If the runtime of point proposals is predictable we suggest our new `rambo` approach for parallel MBO with high numbers of available CPUs. Even if the runtime estimation quality is obviously hard to determine in advance, for real applications like hyperparameter optimization for machine learning methods predictable runtimes can be assumed. Our results also suggest that on some setups the choice of the infill criterion determines which parallelization strategy can reach a better performance. For future work a criterion that assigns an infill value to a set of candidates that can be scheduled without causing long idle times appears promising. Furthermore we want to include the memory consumption measured by the `traceR` [16,17] tool into our scheduling decisions for experiments with high memory demands.

Acknowledgments. J. Richter and H. Kotthaus — These authors contributed equally. This work was partly supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876, A3 and by Competence Network for Technical, Scientific High Performance Computing in Bavaria (KONWIHR) in the project “Implementierung und Evaluation eines Verfahrens zur automatischen, massivparallelen Modellselektion im Maschinellen Lernen”.

References

1. Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., Tierney, K.: Model-based genetic algorithms for algorithm configuration. In: International Joint Conference on Artificial Intelligence, pp. 733–739 (2015)
2. Bischl, B., Lang, M., Kotthoff, L., Schiffner, J., Richter, J., Studerus, E., Casalicchio, G., Jones, Z.M.: mlr: Machine learning in R. J. Mach. Learn. Res. **17**(170), 1–5 (2016)

3. Bischl, B., Richter, J., Bossek, J., Horn, D., Thomas, J., Lang, M.: mlrMBO: A modular framework for model-based optimization of expensive black-box functions. arXiv pre-print (2017). http://arxiv.org/abs/1703.03373
4. Bischl, B., Wessing, S., Bauer, N., Friedrichs, K., Weihs, C.: MOI-MBO: multi-objective infill for parallel model-based optimization. In: Pardalos, P.M., Resende, M.G.C., Vogiatzis, C., Walteros, J.L. (eds.) LION 2014. LNCS, vol. 8426, pp. 173–186. Springer, Cham (2014). doi:10.1007/978-3-319-09584-4_17
5. Borchers, H.: adagio: Discrete and Global Optimization Routines (2016). R package version 0.6.5. https://CRAN.R-project.org/package=adagio
6. Bossek, J.: smoof: Single and Multi-Objective Optimization Test Functions (2016). R package version 1.4. https://CRAN.R-project.org/package=smoof
7. Chevalier, C., Ginsbourger, D.: Fast computation of the multi-points expected improvement with applications in batch selection. In: Nicosia, G., Pardalos, P. (eds.) LION 2013. LNCS, vol. 7997, pp. 59–69. Springer, Heidelberg (2013). doi:10.1007/978-3-642-44973-4_7
8. Ginsbourger, D., Janusevskis, J., Le Riche, R.: Dealing with asynchronicity in parallel Gaussian process based global optimization. In: 4th International Conference of the ERCIM WG on Computing & Statistics (ERCIM 2011), pp. 1–27 (2011)
9. Ginsbourger, D., Le Riche, R., Carraro, L.: Kriging is well-suited to parallelize optimization. In: Tenne, Y., Goh, C.K. (eds.) Computational Intelligence in Expensive Optimization Problems, pp. 131–162. Springer, Heidelberg (2010). doi:10.1007/978-3-642-10701-6_6
10. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25566-3_40
11. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Parallel algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, pp. 55–70. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34413-8_5
12. Janusevskis, J., Le Riche, R., Ginsbourger, D.: Parallel expected improvements for global optimization: summary, bounds and speed-up. Technical report (2011). https://hal.archives-ouvertes.fr/hal-00613971
13. Janusevskis, J., Le Riche, R., Ginsbourger, D., Girdziusas, R.: Expected improvements for the asynchronous parallel global optimization of expensive functions: potentials and challenges. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, pp. 413–418. Springer, Heidelberg (2012). doi:10.1007/978-3-642-34413-8_37
14. Jones, D.R.: A taxonomy of global optimization methods based on response surfaces. J. Global Optim. **21**(4), 345–383 (2001)
15. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. J. Global Optim. **13**(4), 455–492 (1998)
16. Kotthaus, H., Korb, I., Lang, M., Bischl, B., Rahnenführer, J., Marwedel, P.: Runtime and memory consumption analyses for machine learning R programs. J. Stat. Comput. Simul. **85**(1), 14–29 (2015)
17. Kotthaus, H., Korb, I., Marwedel, P.: Performance analysis for parallel R programs: towards efficient resource utilization. Technical report 01/2015, Department of Computer Science 12, TU Dortmund University (2015)
18. Lang, M., Bischl, B., Surmann, D.: batchtools: Tools for R to work on batch systems. J. Open Source Softw. **2**(10) (2017)

19. Richter, J., Kotthaus, H., Bischl, B., Marwedel, P., Rahnenführer, J., Lang, M.: Faster model-based optimization through resource-aware scheduling strategies. In: Festa, P., Sellmann, M., Vanschoren, J. (eds.) LION 2016. LNCS, vol. 10079, pp. 267–273. Springer, Cham (2016). doi:10.1007/978-3-319-50349-3_22
20. Roustant, O., Ginsbourger, D., Deville, Y.: DiceKriging, DiceOptim: two R packages for the analysis of computer experiments by Kriging-based metamodeling and optimization. J. Stat. Softw. **51**(1), 1–55 (2012)
21. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems, vol. 25. pp. 2951–2959. Curran Associates, Inc. (2012)
22. Strongin, R.G., Sergeyev, Y.D.: Global Optimization with Non-Convex Constraints. Kluwer Academic Publishers, Dordrecht (2000)
23. Zhigljavsky, A., Žilinskas, A.: Stochastic Global Optimization. Springer, New York (2008). doi:10.1007/978-0-387-74740-8

7. Gradient Boosting for Distributional Regression: Faster Tuning and Improved Variable Selection via Noncyclical Updates

Declaration of the specific contributions of the author This publication is based on the author's master thesis *Stability selection for component-wise gradient boosting*. For the publication the text was shortened and rewritten by the author and refined by all co-authors. Additional experiments for the out-of-bag error performance have been conducted by the author, as well as new and improved figures. The implementation of the new algorithm in the `gamboostLSS` package was done by the author and guided and supported by Benjamin Hofner and Andreas Mayr. The idea for a noncyclical fitting algorithm was proposed by Matthias Schmid, Benjamin Hofner, and Andreas Mayr, they also contributed to its further methodological development. Adam Smith provided domain knowledge for the use-case and Matthias Schmid helped with the motivation and general context of the research domain. Bernd Bischl gave valuable insights for the benchmarks studies.

Stat Comput (2018) 28:673–687
https://doi.org/10.1007/s11222-017-9754-6

Gradient boosting for distributional regression: faster tuning and improved variable selection via noncyclical updates

Janek Thomas[1] · **Andreas Mayr**[2,3] · **Bernd Bischl**[1] · **Matthias Schmid**[3] · **Adam Smith**[4] · **Benjamin Hofner**[5]

Received: 3 November 2016 / Accepted: 5 May 2017 / Published online: 11 May 2017

Abstract We present a new algorithm for boosting generalized additive models for location, scale and shape (GAMLSS) that allows to incorporate stability selection, an increasingly popular way to obtain stable sets of covariates while controlling the per-family error rate. The model is fitted repeatedly to subsampled data, and variables with high selection frequencies are extracted. To apply stability selection to boosted GAMLSS, we develop a new "noncyclical" fitting algorithm that incorporates an additional selection step of the best-fitting distribution parameter in each iteration. This new algorithm has the additional advantage that optimizing the tuning parameters of boosting is reduced from a multi-dimensional to a one-dimensional problem with vastly decreased complexity. The performance of the novel algorithm is evaluated in an extensive simulation study. We apply this new algorithm to a study to estimate abundance of common eider in Massachusetts, USA, featuring excess zeros, overdispersion, nonlinearity and spatiotemporal structures. Eider abundance is estimated via boosted GAMLSS, allowing both mean and overdispersion to be regressed on covariates. Stability selection is used to obtain a sparse set of stable predictors.

Keywords Boosting · Additive models · GAMLSS · GamboostLSS · Stability selection

1 Introduction

In view of the growing size and complexity of modern databases, statistical modeling is increasingly faced with heteroscedasticity issues and a large number of available modeling options. In ecology, for example, it is often observed that outcome variables do not only show differences in *mean* conditions but also tend to be highly *variable* across different geographical features or states of a combination of covariates (e.g., Osorio and Galiano 2012). In addition, ecological databases typically contain large numbers of correlated predictor variables that need to be carefully chosen for possible incorporation in a statistical regression model (Aho et al. 2014; Dormann et al. 2013; Murtaugh 2009.

A convenient approach to address both heteroscedasticity and variable selection in statistical regression models is the combination of GAMLSS modeling with gradient boosting algorithms. GAMLSS, which refer to "generalized additive models for location, scale and shape" (Rigby and Stasinopoulos 2005), are a modeling technique that relates not only the mean but all parameters of the outcome distribution to the available covariates. Consequently, GAMLSS simultaneously fit different submodels for the location, scale and shape parameters of the conditional distribution. Gradient boosting, on the other hand, has become a popular tool for data-driven variable selection in generalized additive models

Electronic supplementary material The online version of this article (doi:10.1007/s11222-017-9754-6) contains supplementary material, which is available to authorized users.

Janek Thomas
janek.thomas@stat.uni-muenchen.de

1 Department of Statistics, Ludwig-Maximilians-Universität München, Ludwigstrasse 33, 80539 Munich, Germany

2 Department of Medical Informatics, Biometry and Epidemiology, FAU Erlangen-Nürnberg, Erlangen, Germany

3 Department of Medical Biometry, Informatics and Epidemiology, University Hospital Bonn, Bonn, Germany

4 U.S. Fish & Wildlife Service, National Wildlife Refuge System, Southeast Inventory & Monitoring Branch, Lewistown, MT, USA

5 Section Biostatistics, Paul-Ehrlich-Institute, Langen, Germany

(Bühlmann and Hothorn 2007). The most important feature of gradient boosting is the ability of the algorithm to perform variable selection in each iteration, so that model fitting and variable selection are accomplished in a single algorithmic procedure. To combine GAMLSS with gradient boosting, we have developed the gamboostLSS algorithm (Mayr et al. 2012) and have implemented this procedure in the R add-on package **gamboostLSS** (Hofner et al. 2016, 2017).

A remaining problem of gradient boosting is the tendency of boosting algorithms to select a relatively high number of false-positive variables and to include too many noninformative covariates in a statistical regression model. This issue, which has been raised in several previous articles (Bühlmann and Hothorn 2010; Bühlmann and Yu 2006; Huang et al. 2012), is particularly relevant for model building in the GAMLSS framework, as the inclusion of noninformative false positives in the submodels for the scale and shape parameters may result in overfitting with a highly inflated variance. As a consequence, it is crucial to include only those covariates in these submodels that show a relevant effect on the outcome parameter of interest. From an algorithmic point of view, this problem is aggravated by the conventional fitting procedure of gamboostLSS: Although the fitting procedure proposed in Mayr et al. (2012) incorporates different iteration numbers for each of the involved submodels, the algorithm starts with mandatory updates of each submodel at the beginning of the procedure. Consequently, due to the tendency of gradient boosting to include relatively high numbers of noninformative covariates, false-positive variables may enter a GAMLSS submodel at a very early stage, even before the iteration number of the submodel is finally reached.

To address these issues and to enforce sparsity in GAMLSS, we propose a novel procedure that incorporates *stability selection* (Meinshausen and Bühlmann 2010) in gamboostLSS. Stability selection is a generic method that investigates the importance of covariates in a statistical model by repeatedly subsampling the data. Sparsity is enforced by including only the most "stable" covariates, in the final statistical model. Importantly, under appropriate regularity conditions, stability selection can be tuned such that the expected number of false-positive covariates is controlled in a mathematically rigorous way. As will be demonstrated in Sect. 3 of this paper, the same property holds in the gamboostLSS framework.

To combine gamboostLSS with stability selection, we present an improved "*noncyclical*" fitting procedure for gamboostLSS that addresses the problem of possible false-positive inclusions at early stages of the algorithm. In contrast to the original "*cyclical*" gamboostLSS algorithm presented in Mayr et al. (2012), the new version of gamboostLSS not only performs variable selection in each iteration but additionally an iteration-wise selection of the best submodel (location, scale, or shape) that leads to the largest improvement in model fit. As a consequence, sparsity is not only enforced by the inclusion of the most "stable" covariates in the GAMLSS submodels but also by a data-driven choice of iteration-wise submodel updates. It is this procedure that theoretically justifies and thus enables the use of stability selection in gamboostLSS.

A further advantage of "*noncyclical*" fitting is that the maximum number of boosting iterations for each submodel does not have to be specified individually for each submodel (as in the originally proposed "*cyclical*" variant), instead only the overall number of iterations must be chosen optimally. Tuning the complete model reduces from a multidimensional to a one-dimensional optimization problem, regardless of the number of submodels, therefore drastically reducing the amount of needed runtime for model selection.

A similar approach for noncyclical fitting of multiparameter models was recently suggested by Messner et al. (2017) for the specific application of ensemble post-processing for weather forecasts. Our proposed method generalizes this approach, allowing gamboostLSS to be combined with stability selection in a generic way that applies to a large number of outcome distributions.

The rest of this paper is organized as follows: In Sect. 2, we describe the gradient boosting, GAMLSS and stability selection techniques and show how to combine the three approaches in a single algorithm. In addition, we provide details on the new gamboostLSS fitting procedure. Results of extensive simulation studies are presented in Sect. 3. They demonstrate that combining gamboostLSS with stability selection results in prediction models that are both easy to interpret and show a favorable behavior with regard to variable selection. They also show that the new gamboostLSS fitting procedure results in a large decrease in runtime while showing similar empirical convergence rates as the traditional gamboostLSS procedure. We present an application of the proposed algorithm to a spatiotemporal data set on sea duck abundance in Nantucket Sound, USA, in Sect. 4. Section 5 summarizes the main findings and provides details on the implementation of the proposed methodology in the R package **gamboostLSS** (Hofner et al. 2017).

2 Methods

2.1 Gradient boosting

Gradient boosting is a supervised learning technique that combines an ensemble of *base-learners* to estimate complex statistical dependencies. Base-learners should be *weak* in the sense that they only possess moderate prediction accuracy, usually assumed to be at least slightly better than a random predictor, but on the other hand easy and fast to fit. Base-learners can be, for example, simple linear regression models, regression splines with low degrees of freedom, or stumps

Stat Comput (2018) 28:673–687 675

(i.e., trees with only one split; Bühlmann and Hothorn 2007). One base-learner by itself will usually not be enough to fit a well-performing statistical model to the data, but a boosted combination of a large number can compete with other state-of-the-art algorithms on many tasks, e.g., classification (Li 2012) or image recognition (Opelt et al. 2004).

Let $D = \{(x^{(i)}, y^{(i)})\}_{i=1,\ldots,n}$ be a learning data set sampled i.i.d. from a distribution over the joint space $\mathcal{X} \times \mathcal{Y}$, with a p-dimensional input space $\mathcal{X} = (\mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_p)$ and a usually one-dimensional output space $\mathcal{Y}$. The response variable is estimated through an additive model where $\mathbb{E}(y|x) = g^{-1}(\eta(x))$, with link function g and additive predictor $\eta : \mathcal{X} \to \mathbb{R}$,

$$\eta(x) = \beta_0 + \sum_{j=1}^{J} f_j(x|\beta_j), \tag{1}$$

with a constant intercept coefficient β_0 and additive effects $f_j(x|\beta_j)$ derived from the pre-defined set of base-learners. These are usually (semi-)parametric effects, e.g., splines, with parameter vector β_j. Note that some effects may later be estimated as 0, i.e., $f_j(x|\beta_j) = 0$. In many cases, each base-learner is defined on exactly one element $\mathcal{X}_j$ of $\mathcal{X}$ and thus Eq. 1 simplifies to

$$\eta(x) = \beta_0 + \sum_{j=1}^{p} f_j(x_j|\beta_j). \tag{2}$$

To estimate the parameters $\beta_1, \ldots, \beta_J$ of the additive predictor, the boosting algorithm minimizes the *empirical risk* R which is the loss $\rho : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$ summed over all training data:

$$R = \sum_{i=1}^{n} \rho(y^{(i)}, \eta(x^{(i)})). \tag{3}$$

The loss function measures the discrepancy between the true outcome $y^{(i)}$ and the additive predictor $\eta(x^{(i)})$. Examples are the absolute loss $|y^{(i)} - \eta(x^{(i)})|$, leading to a regression model for the median, the quadratic loss $(y^{(i)} - \eta(x^{(i)}))^2$, leading to the conventional (mean) regression model or the binomial loss $-y^{(i)}\eta(x^{(i)}) + \log(1 + \exp(\eta(x^{(i)})))$ often used in classification of binary outcomes $y^{(i)} \in \{0, 1\}$. Very often the loss is derived from the negative log likelihood of the distribution of $\mathcal{Y}$, depending on the desired model (Friedman et al. 2000).

While there exist different types of gradient boosting algorithms (Mayr et al. 2014a, b), in this article we will focus on component-wise gradient boosting (Bühlmann and Yu 2003; Bühlmann and Hothorn 2007). The main idea is to fit simple regression-type base-learners $h(\cdot)$ one by one to the negative gradient vector of the loss $u = (u^{(1)}, \ldots, u^{(n)})$ instead of to the true outcomes $y = (y^{(1)}, \ldots, y^{(n)})$. Base-learners are chosen in such a way that they approximate the effect $\hat{f}(x|\beta_j) = \sum_m h_j(\cdot)$. The negative gradient vector in iteration m, evaluated at the estimated additive predictor $\hat{\eta}^{[m-1]}(x^{(i)})$, is defined as

$$\boldsymbol{u} = \left(-\frac{\partial}{\partial \eta} \rho(y, \eta) \Big|_{\eta = \hat{\eta}^{[m-1]}(x^{(i)}),\, y = y^{(i)}} \right)_{i=1,\ldots,n}.$$

In every boosting iteration, each base-learner is fitted separately to the negative gradient vector by least-squares or penalized least-squares regression. The best-fitting base-learner is selected based on the residual sum of squares with respect to u

$$j^* = \underset{j \in 1 \ldots J}{\operatorname{argmin}} \sum_{i=1}^{n} (u^{(i)} - \hat{h}_j(x^{(i)}))^2. \tag{4}$$

Only the best-performing base-learner $\hat{h}_{j^*}(x)$ will be used to update the current additive predictor,

$$\hat{\eta}^{[m]} = \hat{\eta}^{[m-1]} + \mathrm{sl} \cdot \hat{h}_{j^*}(x) \tag{5}$$

where $0 < \mathrm{sl} \ll 1$ denotes the step length (learning rate; usually $\mathrm{sl} = 0.1$). The choice of sl is not of critical importance as long as it is sufficiently small (Schmid and Hothorn 2008).

The main tuning parameter for gradient boosting algorithms is the number of iterations m that are performed before the algorithm is stopped (denoted as m_{stop}). The selection of m_{stop} has a crucial influence on the prediction performance of the model. If m_{stop} is set too small, the model cannot fully incorporate the influence of the effects on the response and will consequently have a poor performance. On the other hand, too many iterations will result in *overfitting*, which hampers the interpretation and generalizability of the model.

2.2 GAMLSS

In classical generalized additive models (GAM, Hastie and Tibshirani 1990), it is assumed that the conditional distribution of $\mathcal{Y}$ depends only on one parameter, usually the conditional mean. If the distribution has multiple parameters, all but one are considered to be constant nuisance parameters. This assumption will not always hold and should be critically examined, e.g., the assumption of constant variance is not adequate for heteroscedastic data. Potential dependency of the scale (and shape) parameter(s) of a distribution on predictors can be modeled in a similar way to the conditional mean (i.e., location parameter). This extended model class is called *generalized additive models for location, scale and shape* (GAMLSS, Rigby and Stasinopoulos 2005).

The framework hence fits different prediction functions to multiple distribution parameters $\theta = (\theta_1, \ldots, \theta_k), k = 1, \ldots, 4$. Given a conditional density $p(y|\theta)$, one estimates additive predictors (cf. Eq. 1) for each of the parameters θ_k

$$\eta_{\theta_k} = \beta_{0\theta_k} + \sum_{j=1}^{J_k} f_{j\theta_k}(x|\beta_{j\theta_k}), \qquad k = 1, \ldots, 4. \tag{6}$$

Typically, these models are estimated via penalized likelihood. For details on the fitting algorithm, see Rigby et al. (2008).

Even though these models can be applied to a large number of different situations, and the available fitting algorithms are extremely powerful, they still inherit some shortcomings from the penalized likelihood approach:

1. It is not possible to estimate models with more covariates than observations.
2. Maximum likelihood estimation does not feature an embedded variable selection procedure. For GAMLSS models, the standard AIC has been expanded to the *generalized* AIC (GAIC) in Rigby and Stasinopoulos (2005) to be applied to multi-dimensional prediction functions. Variable selection via information criteria has several shortcomings, for example the inclusion of too many non-informative variables (Anderson and Burnham 2002).
3. Whether to model predictors in a linear or nonlinear fashion is not trivial. Unnecessary complexity increases the danger of overfitting as well as computation time. Again, a generalized criterion like GAIC must be used to choose between linear and nonlinear terms.

2.3 Boosted GAMLSS

To deal with these issues, gradient boosting can be used to fit the model instead of the standard maximum likelihood algorithm. Based on an approach proposed in Schmid et al. (2010) to fit zero-inflated count models, in Mayr et al. (2012) the author developed a general algorithm to fit multi-dimensional prediction functions with component-wise gradient boosting (see Algorithm 1).

The basic idea is to cycle through the distribution parameters θ in the fitting process. Partial derivatives with respect to each of the additive predictors are used as response vectors. In each iteration of the algorithm, the best-fitting base-learner is determined for *each* distribution parameter, while all other parameters stay fixed. For a four parametric distribution, the update in boosting iteration $m+1$ may be sketched as follows:

$$\frac{\partial}{\partial \eta_{\theta_1}} \rho(y, \hat{\theta}_1^{[m]}, \hat{\theta}_2^{[m]}, \hat{\theta}_3^{[m]}, \hat{\theta}_4^{[m]}) \xrightarrow{\text{update}} \eta_{\theta_1}^{[m+1]}$$

$$\frac{\partial}{\partial \eta_{\theta_2}} \rho(y, \hat{\theta}_1^{[m+1]}, \hat{\theta}_2^{[m]}, \hat{\theta}_3^{[m]}, \hat{\theta}_4^{[m]}) \xrightarrow{\text{update}} \eta_{\theta_2}^{[m+1]}$$

$$\frac{\partial}{\partial \eta_{\theta_3}} \rho(y, \hat{\theta}_1^{[m+1]}, \hat{\theta}_2^{[m+1]}, \hat{\theta}_3^{[m]}, \hat{\theta}_4^{[m]}) \xrightarrow{\text{update}} \eta_{\theta_3}^{[m+1]}$$

$$\frac{\partial}{\partial \eta_{\theta_4}} \rho(y, \hat{\theta}_1^{[m+1]}, \hat{\theta}_2^{[m+1]}, \hat{\theta}_3^{[m+1]}, \hat{\theta}_4^{[m]}) \xrightarrow{\text{update}} \eta_{\theta_4}^{[m+1]}.$$

Unfortunately, separate stopping values for each distribution parameter have to be specified, as the prediction functions will most likely require different levels of complexity and hence a different number of boosting iterations. In case of multi-dimensional boosting, the different $m_{\text{stop},k}$ values are not independent of each other and have to be jointly optimized. The usually applied *grid search* scales exponentially with the number of distribution parameters and can quickly become computationally demanding or even infeasible.

Algorithm 1 "Cyclical" component-wise gradient boosting in multiple dimensions (Mayr et al. 2012)

Initialize

1. Initialize the additive predictors $\hat{\eta}^{[0]} = (\hat{\eta}_{\theta_1}^{[0]}, \hat{\eta}_{\theta_2}^{[0]}, \hat{\eta}_{\theta_3}^{[0]}, \hat{\eta}_{\theta_4}^{[0]})$ with offset values.
2. For each distribution parameter $\theta_k, k = 1, \ldots, 4$, specify a set of base-learners, i.e., for parameter θ_k define $h_{k1}(x^{(i)}), \ldots, h_{kJ_k}(x^{(i)})$ where J_k is the cardinality of the set of base-learners specified for θ_k.

Boosting in multiple dimensions

For $m = 1$ to $\max(m_{\text{stop},1}, \ldots, m_{\text{stop},4})$:

3. For $k = 1$ to 4:
   (a) **If** $m > m_{\text{stop},k}$ set $\hat{\eta}_{\theta_k}^{[m]} := \hat{\eta}_{\theta_k}^{[m-1]}$ and skip this iteration.
   Else compute negative partial derivative $-\frac{\partial}{\partial \eta_{\theta_k}} \rho(y, \eta)$ an plug in the current estimates $\hat{\eta}^{[m-1]}(\cdot)$:
   $$u_k = \left(\frac{\partial}{\partial \eta_{\theta_k}} \rho(y, \eta) \Big|_{\eta = \hat{\eta}^{[m-1]}(x^{(i)}), y = y^{(i)}} \right)_{i=1,\ldots,n}$$
   (b) **Fit** each of the base-learners u_k contained in the set of base-learners specified for the distribution parameter θ_k in step (2) to the negative gradient vector.
   (c) **Select** the component j^* that best fits the negative partial derivative vector according to the residual sum of squares, i.e., select the base-learner h_{kj^*} defined by
   $$j^* = \underset{j \in 1,\ldots,J_k}{\operatorname{argmin}} \sum_{i=1}^{n} (u_k^{(i)} - \hat{h}_{kj}(x^{(i)}))^2.$$
   (d) **Update** the additive predictor η_{θ_k}
   $$\hat{\eta}_{\theta_k}^{[m]} = \hat{\eta}_{\theta_k}^{[m-1]} + \text{sl} \cdot \hat{h}_{kj^*}(x),$$
   where sl is the step length (typically $\text{sl} = 0.1$), and update the current estimates for step 4(a):
   $$\hat{\eta}_{\theta_k}^{[m-1]} = \hat{\eta}_{\theta_k}^{[m]}.$$

Stat Comput (2018) 28:673–687 677

2.4 Stability selection

Selecting an optimal subset of explanatory variables is a crucial step in almost every supervised data analysis problem. Especially in situations with a large number of covariates, it is often almost impossible to get meaningful results without *automatic*, or at least *semiautomatic*, selection of the most relevant predictors. Selection of covariate subsets based on modified R^2 criteria (e.g., the AIC) can be unstable, see, for example, Flack and Chang (1987), and tend to select too many covariates (see, e.g., Mayr et al. 2012).

Component-wise boosting algorithms are one solution to select predictors in high dimensions and/or $p > n$ problems. As only the best-fitting base-learner is selected to update the model in each boosting step, as discussed above, variable selection can be obtained by stopping the algorithm early enough. Usually, this is done via cross-validation methods, selecting the stopping iteration that optimizes the empirical risk on test data (*predictive* risk). Hence, boosting with *early stopping* via cross-validation offers a way to perform variable selection while fitting the model. Nonetheless, boosted models stopped early via cross-validation still have a tendency to include too many noise variables, particularly in rather low-dimensional settings with few possible predictors and many observations ($n > p$; Bühlmann et al. 2014).

2.4.1 Stability selection for boosted GAM models

To circumvent the problems mentioned above, the *stability selection* approach was developed (Meinshausen and Bühlmann 2010; Shah and Samworth 2013). This generic algorithm can be applied to boosting and all other variable selection methods. The main idea of *stability selection* is to run the selection algorithm on multiple subsamples of the original data. Highly relevant base-learners should be selected in (almost) all subsamples.

Stability selection in combination with boosting was investigated in Hofner et al. (2015) and is outlined in Algorithm 2. In the first step, B random subsets of size $\lfloor n/2 \rfloor$ are drawn, and a boosting model is fitted to each one. The model fit is interrupted as soon as q different base-learners have entered the model. For each base-learner, the selection frequency $\hat{\pi}_j$ is the fraction of subsets in which the base-learner j was selected (7). An effect is included in the model if the selection frequency exceeds the user-specified threshold π_{thr} (8).

This approach leads to upper bounds for the *per-family error rate* (PFER) $\mathbb{E}(V)$, where V is the number of noninformative base-learners wrongly included in the model (i.e., false positives; Meinshausen and Bühlmann 2010):

$$\mathbb{E}(V) \leq \frac{q^2}{(2\pi_{\text{thr}} - 1)p}. \tag{9}$$

Algorithm 2 Stability selection for model-based boosting

1. For $b = 1, \ldots, B$:
   (a) Draw a subset of size $\lfloor n/2 \rfloor$ from the data
   (b) Fit a boosting model until the number of selected base-learners is equal to q or the number of iterations reaches a pre-specified number (m_{stop}).
2. Compute the relative selection frequencies per base-learner:

$$\hat{\pi}_j := \frac{1}{B} \sum_{b=1}^{B} \mathbb{I}_{\{j \in \hat{S}_b\}}, \tag{7}$$

   where $\hat{S}_b$ denotes the set of selected base-learners in iteration b.
3. Select base-learners with a selection frequency of at least π_{thr}, which yields a set of stable covariates

$$\hat{S}_{\text{stable}} := \{j : \hat{\pi}_j \geq \pi_{\text{thr}}\}. \tag{8}$$

Under certain assumptions, refined, less conservative error bounds can be derived (Shah and Samworth 2013).

One of the main difficulties of stability selection in practice is the choice of the parameters q, π_{thr} and PFER. Even though only two of the three parameters need to be specified (the last one can then be derived under equality in (9)), their choice is not trivial and not always intuitive for the user.

Meinshausen and Bühlmann (2010) state that the threshold should be $\pi_{\text{thr}} \in (0.6, 0.9)$ and has little influence on the result. The number of base-learners q has to be sufficiently large, i.e., q should be at least as big as the number of informative variables in the data (or better to say the number of corresponding base-learners). This is obviously a problem in practical applications, in which the number of informative variables (or base-learners) is usually unknown. One nice property is that if q is fixed, π_{thr} and the PFER can be varied without the need to refit the model. A general advice would thus be to choose q relatively large or to make sure that q is large enough for a given combination of π_{thr} and PFER. Simulation studies like Hofner et al. (2015), Mayr et al. (2016) have shown that the PFER is quite conservative and the true number of false positives will most likely be much smaller than the specified value.

In practical applications, two different approaches to select the parameters are typically used. Both assume that the number of covariates to include, q, is chosen intuitively by the user: The first idea is to look at the calculated inclusion frequencies $\hat{\pi}_j$ and look for a *breakpoint* in the decreasing order of the values. The threshold can be then chosen so that all covariates with inclusion frequencies larger than the breakpoint are included, and the resulting PFER is only used as an additional information. The second possibility is to fix the PFER as a conservative upper bound for the expected number of false-positive base-learners. Hofner et al. (2015) provide some rationales for the selection of the PFER by

relating it to common error types, the per-comparison error (i.e., the type I error without multiplicity correction) and the family-wise error rate (i.e., with conservative multiplicity correction).

2.4.2 Stability selection for boosted GAMLSS models

The question of variable selection in (boosted) GAMLSS models is even more critical than in regular (GAM) models, as the question of including a base-learner implies not only if the base-learner should be used in the model at all, but also for which distribution parameter(s) it should be used. Essentially, the number of possible base-learners doubles in a distribution with two parameters, triples in one with three parameters and so on. This is particularly challenging in situations with a large amount of base-learners and in $p > n$ situations.

The method of fitting boosted GAMLSS models in a cyclical way leads to a severe problem when used in combination with stability selection. In each iteration of the algorithm, *all* distribution parameters will receive an additional base-learner as long as their m_{stop} limit is not exceeded. This means that base-learners are added to the model that might have a rather small importance compared to base-learners for other distribution parameters. This becomes especially relevant if the number of informative base-learners varies substantially between distribution parameters.

Regarding the maximum number of base-learners q to be considered in the model, base-learners are counted separately for each distribution parameter, so a base-learner that is selected for the location *and* scale parameter counts as two different base-learners. Arguably, one might circumvent this problem by choosing a higher value for q, but still less stable base-learners could be selected instead of stable ones for other distribution parameters. One aspect of the problem is that the possible model improvement between different distribution parameters is not considered. A careful selection of m_{stop} per distribution parameter might resolve the problem, but the process would still be unstable because the margin of base-learner selection in later stages of the algorithm is quite small. Furthermore, this is not in line with the general approach of stability selection where the standard tuning parameters do not play an important role.

2.5 Noncyclical fitting for boosted GAMLSS

The main idea to solve the previously stated problems of the cyclical fitting approach is to update only one distribution parameter in each iteration, i.e., the distribution parameter with a base-learner that leads to the highest loss reduction over all distribution parameters and base-learners.

Usually, base-learners are selected by comparing their residual sum of squares with respect to the negative gradient vector (*inner loss*). This is done in step (4c) of Algorithm 1 where the different base-learners are compared. However, the residual sum of squares cannot be used to compare the fit of base-learners over different distribution parameters, as the gradients are not comparable.

Inner loss One solution is to compare the empirical risks (i.e., the negative log likelihood of the modeled distribution) after the update with the best-fitting base-learners that have been selected via the residual sum of squares for each distribution parameter: first, for each distribution parameter the best-performing base-learner is selected via the residual sum of squares of the base-learner fit with respect to the gradient vector. Then, the potential improvement in the empirical loss $\Delta\rho$ is compared for all selected base-learners (i.e., over all distribution parameters). Finally, only the best-fitting base-learner (w.r.t. the inner loss) which leads to the highest improvement (w.r.t. the outer loss) is updated. The base-learner selection for each distribution parameter is still done with the *inner loss* (i.e., the residual sum of squares), and this algorithm will be called analogously.

Outer loss Choosing base-learners and parameters with respect to two different optimization criteria may not always lead to the best possible update. A better solution could be to use a criterion which can compare all base-learners for all distribution parameters. As stated, the inner loss cannot be used for such a comparison. However, the empirical loss (i.e., the negative log likelihood of the modeled distribution) can be used to compare both, the base-learners within a distribution parameter and over the different distribution parameters. Now, the negative gradients are used to estimate all base-learners $\hat{h}_{11}, \ldots, \hat{h}_{1p_1}, \hat{h}_{21}, \ldots, \hat{h}_{4p_4}$. The improvement in the empirical risk is then calculated for each base-learner of every distribution parameter, and only the overall best-performing base-learner (w.r.t. the outer loss) is updated. Instead of the using the inner loss, the whole selection process is hence based on the *outer loss* (empirical risk), and the method is named accordingly.

The noncyclical fitting algorithm is shown in Algorithm 3. The *inner* and *outer* variants solely differ in step (3c).

A major advantage of both noncyclical variants compared to the cyclical fitting algorithm (Algorithm 1) is that m_{stop} is always scalar. The updates of each distribution parameter estimate are adaptively chosen. The optimal partitioning (and sequence) of base-learners between different parameters is done automatically while fitting the model. Such a scalar optimization can be done very efficiently using standard cross-validation methods without the need for a multi-dimensional grid search.

3 Simulation study

In a first step, we carry out simulations to evaluate the performance of the new noncyclical fitting algorithms regarding

Stat Comput (2018) 28:673–687 679

Algorithm 3 "Noncyclical" component-wise gradient boosting in multiple dimensions

Initialize

1. Initialize the additive predictors $\hat{\eta}^{[0]} = (\hat{\eta}^{[0]}_{\theta_1}, \hat{\eta}^{[0]}_{\theta_2}, \hat{\eta}^{[0]}_{\theta_3}, \hat{\eta}^{[0]}_{\theta_4})$ with offset values.
2. For each distribution parameter $\theta_k, k = 1, \ldots, 4$, specify a set of base-learners, i.e., for parameter θ_k define $h_{k1}(\cdot), \ldots, h_{kJ_k}(\cdot)$ where J_k is the cardinality of the set of base-learners specified for θ_k.

Boosting in multiple dimensions

For $m = 1$ to m_{stop}:

3. For $k = 1$ to 4:
   (a) Compute negative partial derivatives $-\frac{\partial}{\partial \eta_{\theta_k}} \rho(y, \eta)$ and plug in the current estimates $\hat{\eta}^{[m-1]}(\cdot)$:
   $$u_k = \left(\frac{\partial}{\partial \eta_{\theta_k}} \rho(y, \eta) \Big|_{\eta = \hat{\eta}^{[m-1]}(x^{(i)}), y = y^{(i)}} \right)_{i=1,\ldots,n}$$
   (b) **Fit** each of the base-learners u_k contained in the set of base-learners specified for the distribution parameter θ_k in step (2) to the negative gradient vector.
   (c) **Select** the best-fitting base-learner h_{kj^*} either by
   - the inner loss, i.e., the residual sum of squares of the base-learner fit w.r.t. u_k:
   $$j^* = \underset{j \in 1,\ldots,J_k}{\operatorname{argmin}} \sum_{i=1}^{n} (u_k^{(i)} - \hat{h}_{kj}(x^{(i)}))^2$$
   - the outer loss, i.e., the negative log likelihood of the modeled distribution after the potential update:
   $$j^* = \underset{j \in 1,\ldots,J_k}{\operatorname{argmin}} \sum_{i=1}^{n} \rho \left(y^{(i)}, \hat{\eta}^{[m-1]}_{\theta_k}(x^{(i)}) + \text{sl} \cdot \hat{h}_{kj}(x^{(i)}) \right)$$
   (d) Compute the possible improvement of this update regarding the outer loss
   $$\Delta\rho_k = \sum_{i=1}^{n} \rho \left(y^{(i)}, \hat{\eta}^{[m-1]}_{\theta_k}(x^{(i)}) + \text{sl} \cdot \hat{h}_{kj^*}(x^{(i)}) \right)$$
4. **Update**, depending on the value of the loss reduction $k^* = \text{argmin}_{k \in 1,\ldots,4}(\Delta\rho_k)$ only the overall best-fitting base-learner:
   $$\hat{\eta}^{[m]}_{\theta_{k^*}} = \hat{\eta}^{[m-1]}_{\theta_{k^*}} + \text{sl} \cdot \hat{h}_{k^* j^*}(x)$$
5. Set $\hat{\eta}^{[m]}_{\theta_k} := \hat{\eta}^{[m-1]}_{\theta_k}$ for all $k \neq k^*$.

convergence, convergence speed and runtime. In a second step, we analyze the variable selection properties if the new variant is combined with stability selection.

3.1 Performance of the noncyclical algorithms

The response y_i is drawn from a normal distribution $N(\mu_i, \sigma_i)$, where μ_i and σ_i depend on 4 covariates each. The $x_i, i = 1, \ldots, 6$, are drawn independently from a uniform distribution on $[-1, 1]$, i.e., $n = 500$ samples are drawn independently from $U(-1, 1)$. Two covariates x_3 and x_4 are shared between both μ_i and σ_i, i.e., they are informative for both parameters, which means that there are $p_{\text{inf}} = 6$ informative variables overall. The resulting predictors look like

$$\mu_i = x_{1i} + 2x_{2i} + 0.5x_{3i} - x_{4i}$$
$$\log(\sigma_i) = 0.5x_{3i} + 0.25x_{4i} - 0.25x_{5i} - 0.5x_{6i}.$$

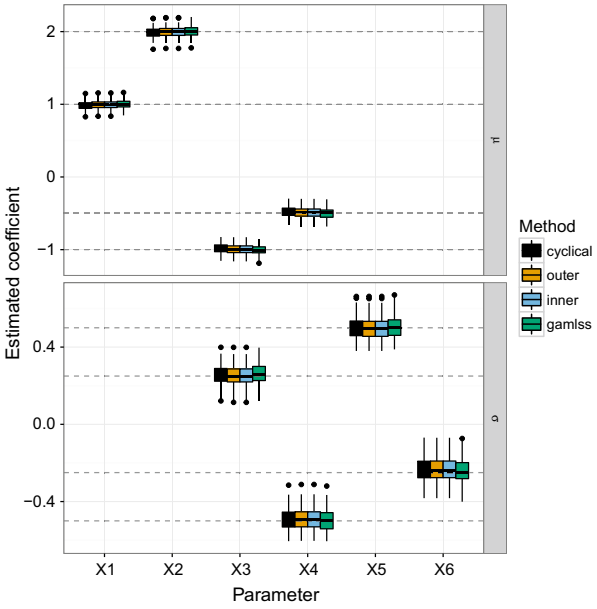

Fig. 1 Distribution of coefficient estimates from $B = 100$ simulation runs. The *dashed lines* show the true parameters. All algorithms were fitted until convergence

Convergence First, we compare the new noncyclical boosting algorithms and the cyclical approach with the classical estimation method based on penalized maximum likelihood (as implemented in the R package **gamlss**, Rigby et al. 2008). The results from $B = 100$ simulation runs are shown in Fig. 1. All four methods converge to the correct solution.

Convergence speed Second, we compare the convergence speed in terms of boosting iterations. Therefore, noninformative variables are added to the model. Four settings are considered with $p_{\text{n-inf}} = 0, 50, 250$ and 500 additional noninformative covariates independently sampled from a $U(-1, 1)$ distribution. With $n = 500$ observations, both $p_{\text{n-inf}} = 250$ and $p_{\text{n-inf}} = 500$ are high-dimensional situations ($p > n$) as we have two distribution parameters. In Fig. 2, the mean risk over 100 simulated data sets is plotted against the number of iterations. The m_{stop} value of the cyclical variant shown in Fig. 2 is the sum of the number of updates on every distribution parameter. Outer and inner loss variants of the noncyclical algorithm have exactly the same risk profiles in all four settings. Compared to the cyclical algorithm, the convergence is faster in the first 500 iterations. After more than 500 iterations, the risk reduction is the

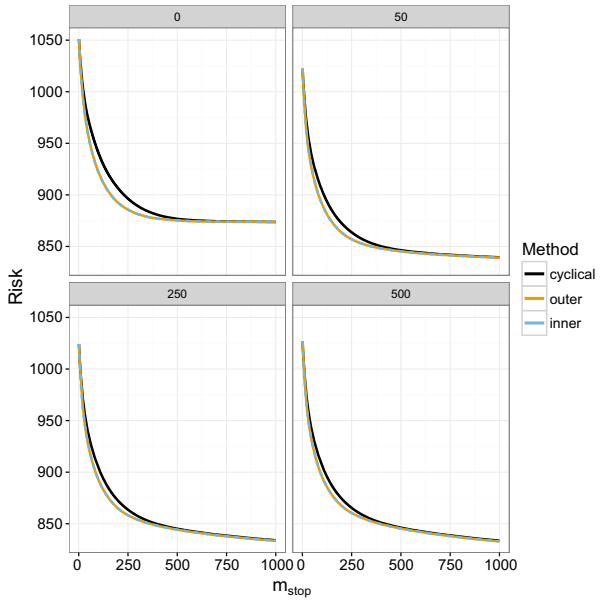

Fig. 2 Convergence speed (regarding the number of boosting iterations m) with 6 informative and $p_{\text{n-inf}} = 0, 50, 250$ and 500 additional noise variables

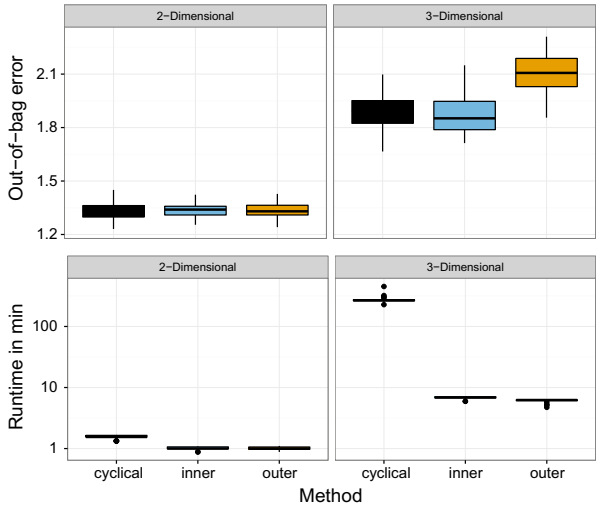

Fig. 3 Out-of-bag error (*top*) and optimization time in minutes (logarithmic scale) Out-of-bag error (*top*) and optimization time in minutes (logarithmic scale; *bottom*) for a two-dimensional (*left*) and three-dimensional distribution (*right*) based on 25-fold bootstrap

same for all three methods. The margin between cyclical and both noncyclical algorithms decreases with a larger number of noise variables.

Runtime The main computational effort of the algorithms is the base-learner selection, which is different for all three methods. The runtime is evaluated in context of cross-validation, which allows us to see how out-of-bag error and runtime behave in different settings. We consider two scenarios—a two-dimensional ($d = 2$) and a three-dimensional ($d = 3$) distribution. The data are generated according to setting 1A and 3A of Sect. 3.2. In each scenario, we sample $n = 500$ observations, but do not add any additional noise variables. For optimization of the model, the out-of-bag prediction error is estimated via a 25-fold bootstrap. A grid of length 10 is created for the cyclical model, with an maximum m_{stop} of 300 for each distribution parameter. The grid is created with the `make.grid` function in **gamboostLSS** (refer to the package documentation for details on the arrangement of the grid points). To allow the same complexity for all variants, the noncyclical methods are allowed up to $m_{\text{stop}} = d \times 300$ iterations.

The results of the benchmark are shown in Fig. 3. The out-of-bag error in the two-dimensional setting is similar for all three methods, but the average number of optimal iterations is considerably smaller for the noncyclical methods (`cyclical`: 360 vs. `inner`: 306, `outer`: 308). In the three-dimensional setting, the outer variant of the noncyclical fitting results in a higher error, whereas the inner variant results in a slightly better performance compared to the cyclical variant. In this setting, the optimal number of iterations is similar for all three methods but near the edge of the searched grid. It is possible that the outer variant will result in a comparable out-of-bag error if the range of the grid is increased.

3.2 Stability selection

After having analyzed the properties of the new noncyclical boosting algorithms for fitting GAMLSS, the remaining question is how they perform when combined with stability selection. In the previous subsection, no differences in the model fit (Fig. 1) and convergence speed (Fig. 2) could be observed, but the optimization results in a three-dimensional setting (Fig. 3) was worse for the outer algorithm. Taking this into consideration, we will only compare the inner and cyclical algorithms here.

We consider three different distributions: (1) the *normal* distribution with two parameters, mean μ_i and standard deviation σ_i. (2) The *negative binomial* distribution with two parameters, mean μ_i and dispersion σ_i. (3) The *zero-inflated negative binomial* (ZINB) distribution with three parameters, μ_i and σ_i identical to the negative binomial distribution, and probability for zero inflation ν_i.

Furthermore, two different partitions of six informative covariates shared between the distribution parameters are evaluated:

(A) *Balanced case* For normal and negative binomial distribution, both μ_i and σ_i depended on four informative covariates, where two are shared. In case of the ZINB distribution, each parameter depends on three informa-

Stat Comput (2018) 28:673–687 681

tive covariates, each sharing one with the other two parameters.

(B) *Unbalanced case* For normal and negative binomial distribution, μ_i depends on five informative covariates, while σ_i only on one. No informative variables are shared between the two parameters. For the ZINB distribution, μ_i depends on five informative variables, σ_i on two, and ν_i on one. One variable is shared across all three parameters.

To summarize these different scenarios for a total of six informative variables, $x_1, \ldots, x_6$:

(1A, 2A)

$$\mu_i = \beta_{1\mu}x_{1i} + \beta_{2\mu}x_{2i} + \beta_{3\mu}x_{3i} + \beta_{4\mu}x_{4i}$$
$$\log(\sigma_i) = \beta_{3\sigma}x_{3i} + \beta_{4\sigma}x_{4i} + \beta_{5\sigma}x_{5i} + \beta_{6\sigma}x_{6i}$$

(1B, 2B)

$$\log(\mu_i) = \beta_{1\mu}x_{1i} + \beta_{2\mu}x_{2i} + \beta_{3\mu}x_{3i} + \beta_{4\mu}x_{4i} + \beta_{5\mu}x_{5i}$$
$$\log(\sigma_i) = \beta_{6\sigma}x_{6i}$$

(3A)

$$\log(\mu_i) = \beta_{1\mu}x_{1i} + \beta_{2\mu}x_{2i} + \beta_{3\mu}x_{3i}$$
$$\log(\sigma_i) = \beta_{3\sigma}x_{3i} + \beta_{4\sigma}x_{4i} + \beta_{5\sigma}x_{5i}$$
$$\text{logit}(\nu_i) = \beta_{1\nu}x_{1i} + \beta_{5\nu}x_{5i} + \beta_{6\nu}x_{6i}$$

(3B)

$$\log(\mu_i) = \beta_{1\mu}x_{1i} + \beta_{2\mu}x_{2i} + \beta_{3\mu}x_{3i} + \beta_{4\mu}x_{4i} + \beta_{5\mu}x_{5i}$$
$$\log(\sigma_i) = \beta_{5\sigma}x_{5i} + \beta_{6\sigma}x_{6i}$$
$$\text{logit}(\nu_i) = \beta_{6\nu}x_{6i}$$

To evaluate the performance of stability selection, two criteria have to be considered. First, the *true-positive rate*, or the number of *true positives* (TP, number of correctly identified informative variable). Secondly, the *false-positive rate*, or the number of *false positives* (FP, number of noninformative variable that were selected as stable predictors).

Considering stability selection, the most obvious control parameter to influence false- and true-positive rates is the threshold π_{thr}. To evaluate the algorithms depending on the settings of stability selection, we consider several values for the number of variables to be included in the model $q \in \{8, 15, 25, 50\}$ and the threshold π_{thr} (varying between 0.55 and 0.99 in steps of 0.01). A third factor is the number of (noise) variables in the model: We consider $p = 50, 250$ or 500 covariates (including the six informative ones). It should be noted that the actual number of possible base-learners is p times the number of distribution parameters, as each covariate can be included in one or more additive predictors. To visualize the simulation results, the progress of true and false positives is plotted against the threshold π_{thr} for different values of p and q, where true and false positives are aggregated over all distribution parameters. Separate figures for each distribution parameter can be found in the web supplement. The setting $p = 50, q = 50$ is an edge case that would work for some assumptions about the distribution of selection probabilities (Shah and Samworth 2013). Since the practical application of this scenario is doubtful, we will not further examine it here.

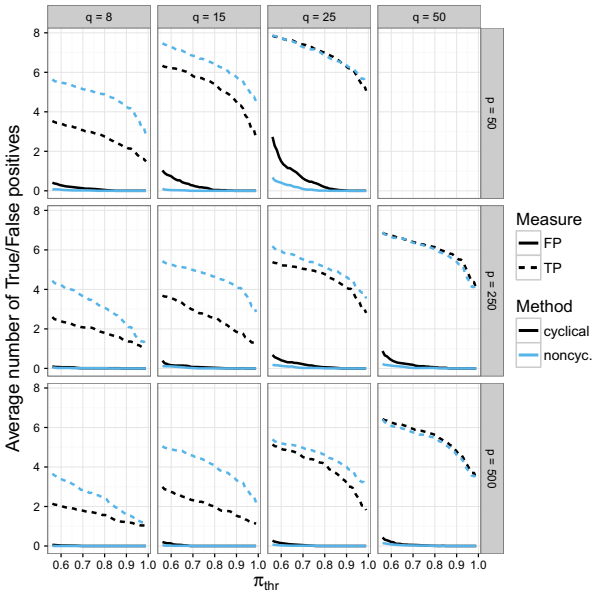

Fig. 4 Balanced case with normal distribution (Scenario 1A)

3.2.1 Results

It can be observed that with increasing threshold π_{thr}, the number of true positives as well as the number of false positives declines in all six scenarios (see Figs. 4, 5, 6, 7, 8, 9) and for every combination of p and q. This is a natural consequence as the threshold is increased, the less variables are selected. Furthermore, the PFER, which is to be controlled by stability selection, decreases with increasing threshold π_{thr} (see Eq. 9).

Results for the normal distribution

In the balanced case (Fig. 4), a higher number of true positives for the noncyclical algorithm can be observed compared to the cyclical algorithm for most simulation settings. Particularly for smaller q values ($q \in \{8, 15\}$), the true-positive rate was always higher compared to the cyclical variant. For

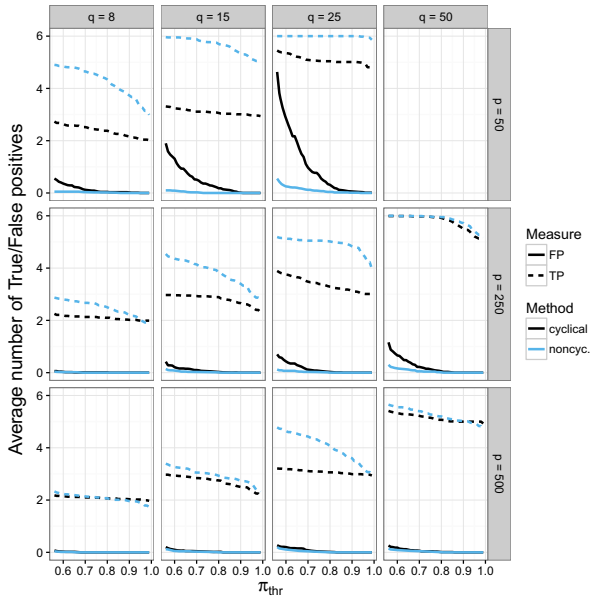

Fig. 5 Unbalanced case with normal distribution (Scenario 1B)

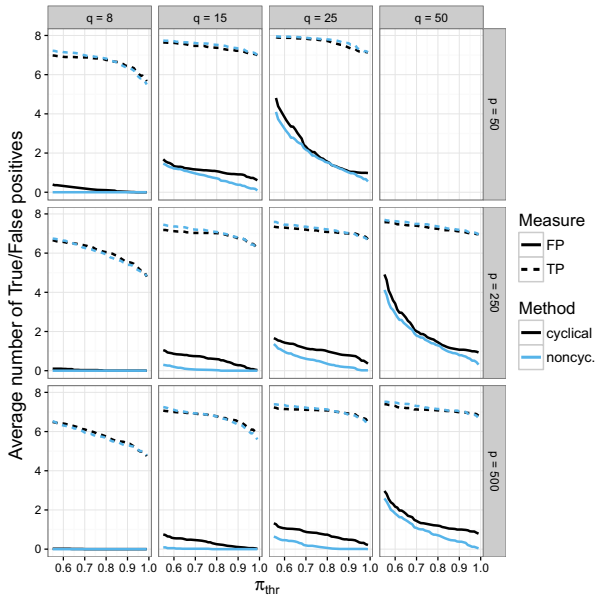

Fig. 6 Balanced case with negative binomial distribution (Scenario 2A)

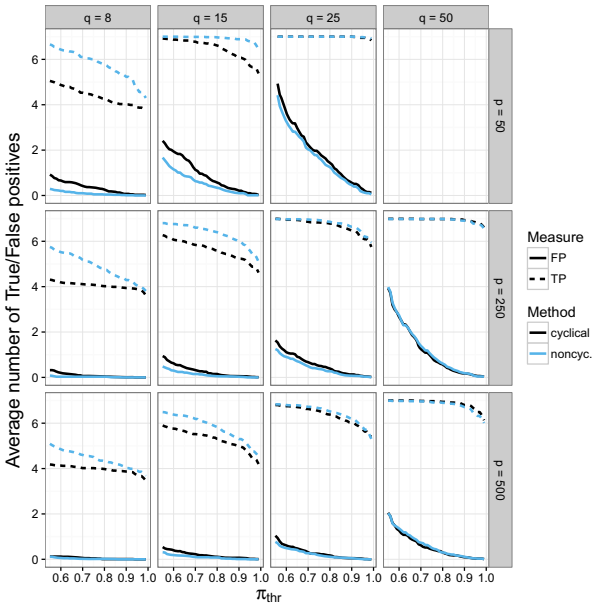

Fig. 7 Unbalanced case with negative binomial distribution (Scenario 2B)

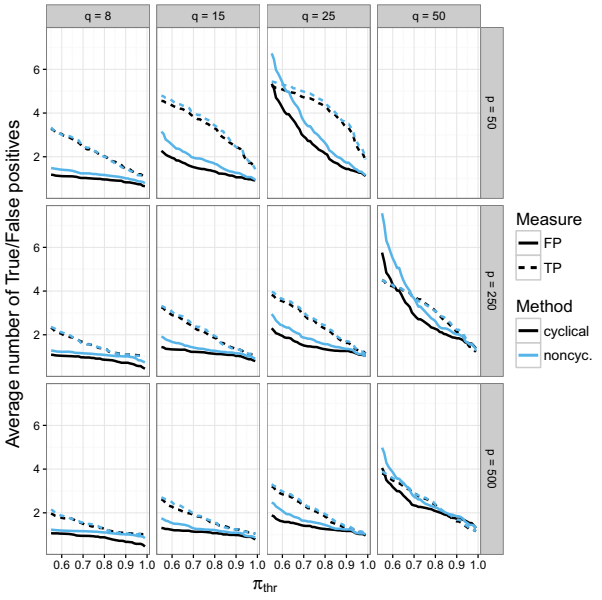

Fig. 8 Balanced case with zero-inflated negative binomial distribution (Scenario 3A)

higher q values, the margin decreases and for the highest settings both methods have approximately the same progression over π_{thr}, with slightly better results for the cyclical algorithm. Overall, the number of true positives increases with a higher value of q. Hofner et al. (2015) found similar results for boosting with one-dimensional prediction functions, but also showed that the true-positive rate decreases again after a certain value of q. This could not be verified for the multi-dimensional case.

The false-positive rate is extremely low for both methods, especially in the high-dimensional settings. The noncyclical fitting method has a constantly smaller or identical false-positive rate and the difference reduces for higher π_{thr}, as expected. For all settings, the false-positive rate reaches zero

Stat Comput (2018) 28:673–687

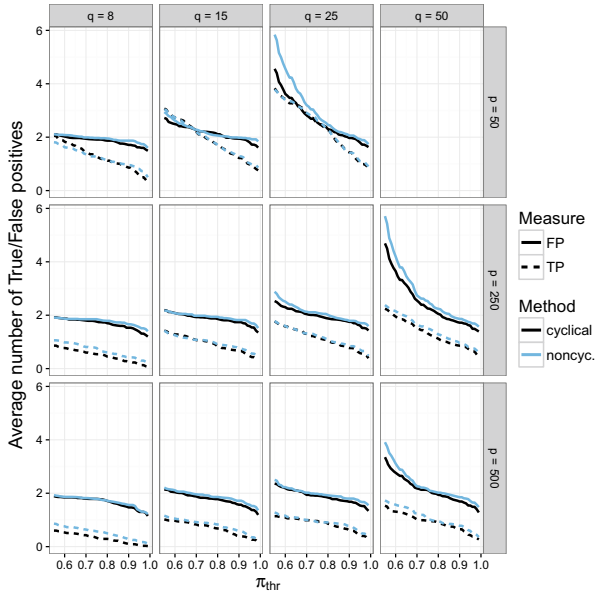

Fig. 9 Unbalanced case with zero-inflated negative binomial distribution (Scenario 3B)

for a threshold higher than 0.9. The setting with the highest false-positive rate is $p = 50$ and $q = 25$, a low-dimensional case with a relatively high threshold. This is also the only setting where on average all 8 informative variables are found (for a threshold of 0.55).

In the unbalanced case (Fig. 5), the results are similar. The number of false positives for the noncyclical variant is lower compared to the cyclical approach in almost all settings. The main difference between the balanced and the unbalanced case is that the number of true positives for the $p = 50, q = 25$ setting is almost identical in the former case, whereas in the latter case the noncyclical variant is dominating the cyclical algorithm. On the other hand, in the high-dimensional case with a small q ($p = 500, q = 8$) both fitting methods have about the same true-positive rate for all possible threshold values.

In summary, it can be seen that the novel noncyclical algorithm is generally better, but at least comparable, to the cyclical method in identifying informative variables. Furthermore, the false-positive rate is less or identical to the cyclical method. For some scenarios in which the scale parameter σ_i is higher compared to the location parameter μ_i, the cyclical variant achieves slightly better results than the noncyclical variant regarding true positives at high p and q values.

Results for the negative binomial distribution

In the balanced case of the negative binomial distribution (Fig. 6), the number of true positives is almost identical for the cyclical and noncyclical algorithm in all settings, while the number of true positives is generally quite high. It varies between 6 and 8 in almost all settings, except for the cases with a very small value of q (=8) where it is slightly lower. This is consistent with the results for stability selection with one-dimensional boosting (Hofner et al. 2015; Mayr et al. 2016). The number of false positives in the noncyclical variants is smaller or identical to the cyclical variant in all tested settings.

In the unbalanced case, the true-positive rate of the noncyclical variant is higher compared to the cyclical variant, whereas the difference reduces for larger values of q. The results are consistent with the normal distribution setting but with smaller differences between both methods.

Results for ZINB distribution

The third considered distribution in our simulation setting is the ZINB distribution, which features three parameters to fit.

In Fig. 8, the results for the balanced case (scenario 3A) are visualized. The tendency of a larger number true positives in the noncyclical variant, which could be observed for both two-parametric distributions, is not present here. For all settings, except for high-dimensional settings with a low q (i.e., $p = 250, 500$ and $q = 50$), the cyclical variant has a higher number of true positives. Additionally, the number of false positives is constantly higher for the noncyclical variant. For the unbalanced setting (Fig. 9), the results are similar in true positives and negatives between both methods.

The number of true positives is overall considerably smaller compared to all other simulation settings. Particularly in the high-dimensional cases ($p = 250, 500$), not even half of the informative covariates are found. In settings with smaller q, the number of true positives is lower than two. Both algorithms obtain approximately the same number of true positives for all settings. In cases with a very low or a very high number q (i.e., $q = 8$ or 50), the noncyclical algorithm is slightly better. The number of false positives is very high, especially compared with the number of true positives and particularly for the unbalanced case. For a lot of settings, more than half of the included variables are noninformative. The number of false positives is higher for the noncyclical case. The difference are especially present in settings with a high q and a low π_{thr}, those settings which also have the highest numbers of true positives.

Altogether, the trend from the simulated two-parameter distributions is not present in the three parametric settings. The cyclical algorithm overall is not worse or even better with regard to both true and false positives for almost all tested scenarios.

4 Modeling sea duck abundance

A recent analysis by Smith et al. (2017) investigated the abundance of wintering sea ducks in Nantucket Sound, Massachusetts, USA. Spatiotemporal abundance data for

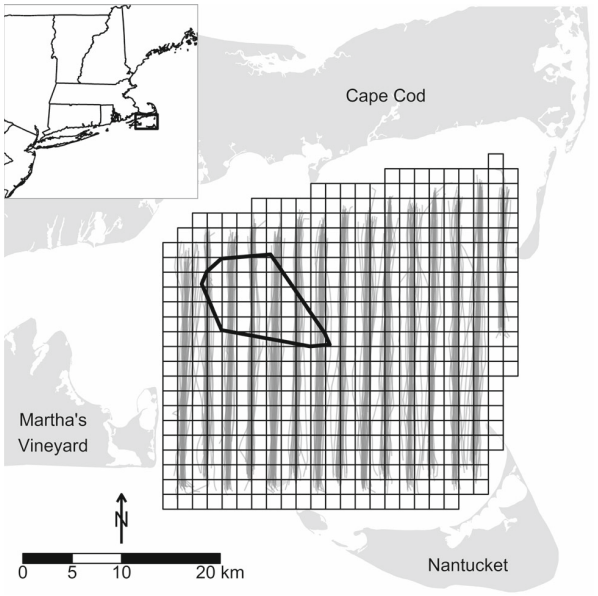

Fig. 10 Nantucket sound—research area of the seabird study by Smith et al. (2017). *Squares* are the discretized segments in which bird abundance was studied. *Gray lines* indicate all aerial transects flown over the course of the study. The *black polygon* indicates the location of permitted wind energy development on Horseshoe Shoal

common eider (among other species) were collected between 2003 and 2005 by counting sea ducks on multiple aerial strip transects from a small plane. For the subsequent analysis, the research area was split in 2.25 km^2 segments (see Fig. 10). Researchers were interested in variables that explained and predicted the distribution of the common eider in the examined area.

As the data were zero-inflated (75% of the segments contained no birds) and highly skewed (a small number of segments contained up to 30,000 birds), a hurdle model (Mullahy 1986) was used for estimation. Therefore, the model was split into an occupancy model (zero part) and an abundance model (count part). The occupancy model estimated if a segment was populated at all and was fitted by boosting a generalized additive model (GAM) with binomial loss, i.e., an additive logistic regression model. In the second step, the number of birds in populated segments was estimated with a boosted GAMLSS model. Because of the skewed and long-tailed data, the (zero-truncated) *negative binomial* distribution was chosen for the abundance model (compare Mullahy 1986).

We reproduce the common eider model reported by Smith et al. (2017) but apply the novel noncyclical algorithm; Smith *et al.* used the cyclic algorithm to fit the GAMLSS model. As discussed in Sect. 3.2, we apply the noncyclical algorithm with inner loss. In short, both distribution parameters, mean *and* overdispersion of the abundance model, and the probability of bird sightings in the occupancy model were regressed on a large number of biophysical covariates, spatial and spatiotemporal effects, and some pre-defined interactions. A complete list of the considered effects can be found in the web supplement. To allow model selection (i.e., the selection between modeling alternatives), the covariate effects were split into linear and nonlinear base-learners (Hothorn et al. 2011; Hofner et al. 2011). The step length was set to sl $= 0.3$, and the optimal number of boosting iterations m_{stop} was found via 25-fold subsampling with sample size $n/2$ (Mayr et al. 2012). Additionally, we used stability selection to obtain sparser models. The numbers of variables to be included per boosting run was set to $q = 35$, and the per-family error rate was set to 6. With unimodality assumption, this resulted in a threshold of $\pi_{\text{thr}} = 0.9$. These settings were chosen identically to the original choices in Smith et al. (2017).

4.1 Results

Subsampling yielded an optimal m_{stop} of 2231, split in $m_{\text{stop},\mu} = 1871$ and $m_{\text{stop},\sigma} = 336$. The resulting model selected 46 out of 48 possible covariates in μ and 8 out of 48 in σ, which is far too complex of a model (especially in μ) to be useful.

With stability selection (see Fig. 12), 10 effects were selected for the location: the intercept, relative sea surface temperature (smooth), chlorophyll a levels (smooth), chromophoric dissolved organic material levels (smooth), sea floor sediment grain size (linear and smooth), sea floor surface area (smooth), mean epibenthic tidal velocity (smooth), a smooth spatial interaction, the presence of nearby ferry routes (yes/no) and two factors to account for changes in 2004 and 2005 compared to the year 2003. For the overdispersion parameter, 5 effects were selected: sea surface temperature (linear), bathymetry (linear), the mean (smooth) and standard deviation (linear) of the epibenthic tidal velocity, and the linear spatial interaction. For the location, all metric variables entered the model nonlinearly. Only sediment grain size was selected linearly as well as nonlinearly in the model. The converse was true for the overdispersion parameter: Only the mean epibenthic velocity was selected as a smooth effect, and all others were selected as linear effects. In Fig. 11, the spatial effects for the mean and overdispersion can be seen. The segment size is based on the spatial covariate with the coarsest resolution, so different resolutions of the segments and their stability were not explored further, though it is likely that this will have some influence on the model performance and the results in Fig. 11.

4.2 Comparison to results of the cyclic method

Comparing the model with the results of Smith et al. (2017), the noncyclical model was larger in μ (10 effects, compared

Stat Comput (2018) 28:673–687

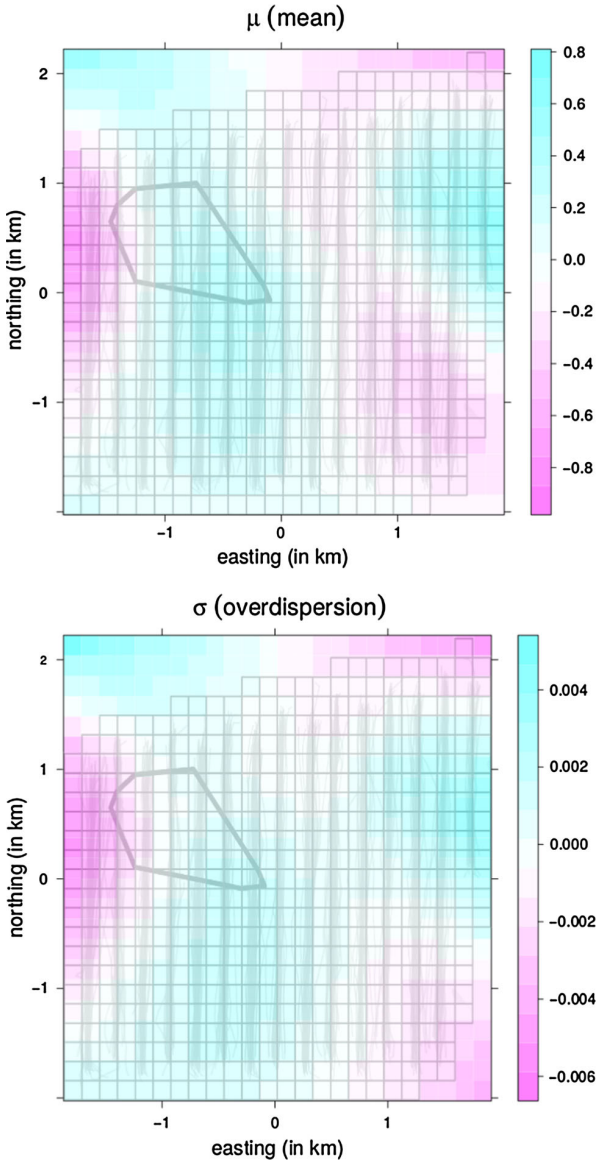

Fig. 11 Spatial effects for mean (*upper figure*) and overdispersion (*lower figure*) of seabird population. The *shaded areas* in the both figures show the research area

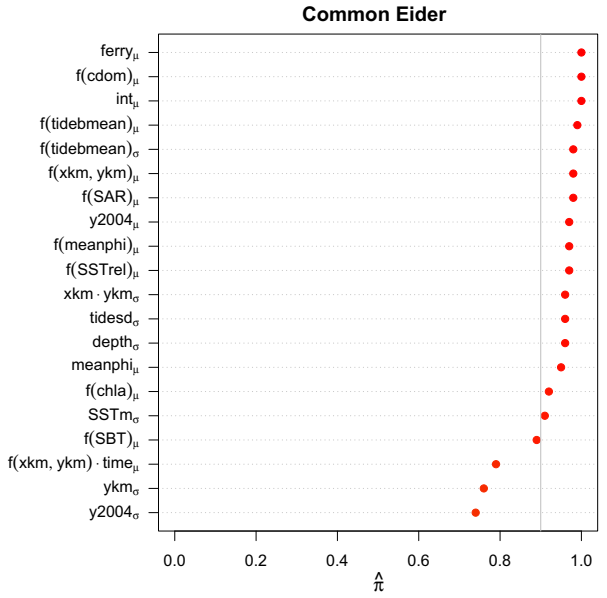

Fig. 12 Selection frequencies of the 20 most frequently selected biophysical covariate base-learners of common eider abundance, determined by stability selection with $q = 35$ and PFER $= 6$. The *gray line* represents the corresponding threshold of 0.9

to 8 effects), but smaller in σ (5 effects, compared to 7 effects). Chlorophyll a levels, mean epibenthic tidal velocity, smooth spatial variation and year were not selected for the mean by stability selection with the cyclical fitting algorithm. On the other hand, bathymetry was selected by the cyclical fitting method, but not by the noncyclical. For the overdispersion parameter, the cyclical algorithm selected the year and the northing of a segment (the north–south position of a segment relative to the median) in addition to all effects selected by the noncyclical variant. Most effects were selected by both the cyclical and the noncyclical algorithm, and the differences in the selected effects were rather small.

In the simulation study for the negative binomial distribution (Sect. 3), the noncyclical variant had a smaller false-positive rate and a higher true-positive rate. Even though the simulation was simplified compared to this application (only linear effects, known true number of informative covariates, uncorrelated effects), the results suggest to prefer the noncyclical variant. Nonetheless, the interpretation of selected covariate effects and final model assessment rests ultimately with subject matter experts.

5 Conclusion

The main contribution of this paper is a statistical model building algorithm that combines the three approaches of gradient boosting, GAMLSS and stability selection. As shown in our simulation studies and the application on sea duck abundance in Sect. 4, the proposed algorithm incorporates the flexibility of structured additive regression modeling via GAMLSS, while it simultaneously allows for a data-driven generation of sparse models.

Being based on the gamboostLSS framework by Mayr et al. (2012), the main feature of the new algorithm is a new "noncyclical" fitting method for boosted GAMLSS models. As shown in the simulation studies, this method does not only increase the flexibility of the variable selection mechanism used in gamboostLSS, but is also more time efficient than the traditional cyclical fitting algorithm. In fact,

even though the initial runtime to fit a single model may be higher (especially if the base-learner selection is done via the outer loss approach), this time is regained while finding the optimal number of boosting iterations via cross-validation approaches. Furthermore, the convergence speed of the new algorithm proved to be faster, and consequently, fewer boosting iterations were needed in total.

Regarding stability selection, we observed that the noncyclical algorithm often had fewer false positives as well as more true positives compared to the cyclical variant in the two-parameter distribution tested in our simulation study. For high-dimensional cases, however, the differences between both methods reduced and, especially with regard to the number of true positives, approximately equal results were achieved. For three-parameter distribution, the cyclical variant achieved better values throughout with respect to both true- and false-positive rates. This may be due to the fact that for more complex distributions, similar densities can be achieved with different parameter settings. For example, in a zero-inflated negative binomial setting, a small location may be hard to distinguish from a large zero inflation. Obviously, the behavior of the cyclical variant is more robust in these situations than the noncyclical variant, which tends to fit very different models on each subsample and consequently selects a higher amount of noninformative variables.

In summary, we have developed a framework for model building in GAMLSS that simplifies traditional optimization approaches to a great extent. For practitioners and applied statisticians, the main consequence of the new methodology is the incorporation of fewer noise variables in the GAMLSS model, leading to sparser and thus more interpretable models. Furthermore, the tuning of the new algorithm is far more efficient and leads to much shorter run times, particularly for complex distributions.

6 Implementation

The derived fitting methods for `gamboostLSS` models are implemented in the R add-on package **gamboostLSS** (Hofner et al. 2017). The fitting algorithm can be specified via the `method` argument. By default, `method` is set to `"cyclical"` which is the originally proposed algorithm. The new inner variant of the noncyclical fitting can be selected with `method = "noncyclic"`. Based on the results of our simulation study, we decided to only support the inner variant in the final package. To ensure reproducibility of the experiments, the state of the package with both inner and outer variants is kept in a separate github branch for this publication, which can be found at http://www.github.com/boost-R/gamboostLSS/tree/stco_paper.

Base-learners and some of the basic methods are implemented in the R package **mboost** (Hothorn et al. 2010; Hofner et al. 2014; Hothorn et al. 2017). The basic fitting algorithm for each distribution parameter is also implemented in **mboost**. For a tutorial and an explanation of technical details of **gamboostLSS**, see Hofner et al. (2016). Stability selection is implemented in the R package **stabs** (Hofner and Hothorn 2017; Hofner et al. 2015), with a specialized function for `gamboostLSS` models, which is included in **gamboostLSS** itself. The development of `mboost`, `gamboostLSS` and `stabs` is hosted openly at

http://www.github.com/boost-R/mboost
http://www.github.com/boost-R/gamboostLSS
http://www.github.com/hofnerb/stabs.

Bug reports and requests should be made there. All packages are also available for installation directly from CRAN.

Acknowledgements We thank Mass Audubon for the use of common eider abundance data.

References

Aho, K., Derryberry, D.W., Peterson, T.: Model selection for ecologists: the worldviews of AIC and BIC. Ecology **95**, 631–636 (2014)
Anderson, D.R., Burnham, K.P.: Avoiding pitfalls when using information-theoretic methods. J. Wildl. Manag. 912–918 (2002)
Bühlmann, P., Hothorn, T.: Boosting algorithms: regularization, prediction and model fitting. Stat. Sci. **22**, 477–505 (2007)
Bühlmann, P., Hothorn, T.: Twin boosting: improved feature selection and prediction. Stat. Comput. **20**, 119–138 (2010)
Bühlmann, P., Yu, B.: Boosting with the L_2 loss: regression and classification. J. Am. Stat. Assoc. **98**, 324–339 (2003)
Bühlmann, P., Yu, B.: Sparse boosting. J. Mach. Learn. Res. **7**, 1001–1024 (2006)
Bühlmann, P., Gertheiss, J., Hieke, S., Kneib, T., Ma, S., Schumacher, M., Tutz, G., Wang, C., Wang, Z., Ziegler, A., et al.: Discussion of "the evolution of boosting algorithms" and "extending statistical boosting". Methods Inf. Med. **53**(6), 436–445 (2014)
Dormann, C.F., Elith, J., Bacher, S., Buchmann, C., Carl, G., Carre, G., Marquez, J.R.G., Gruber, B., Lafourcade, B., Leitao, P.J., Münkemüller, T., McClean, C., Osborne, P.E., Reineking, B., Schröder, B., Skidmore, A.K., Zurell, D., Lautenbach, S.: Collinearity: A review of methods to deal with it and a simulation study evaluating their performance. Ecography **36**, 27–46 (2013)
Flack, V.F., Chang, P.C.: Frequency of selecting noise variables in subset regression analysis: a simulation study. Am. Stat. **41**(1), 84–86 (1987)
Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). Ann. Stat. **28**(2), 337–407 (2000)
Hastie, T.J., Tibshirani, R.J.: Generalized Additive Models, vol. 43. CRC Press, Boca Raton (1990)
Hofner, B., Boccuto, L., Göker, M.: Controlling false discoveries in high-dimensional situations: boosting with stability selection. BMC Bioinf. **16**(1), 144 (2015)
Hofner, B., Hothorn, T.: stabs: stability selection with error control (2017). http://CRAN.R-project.org/package=stabs. R package version 0.6-2
Hofner, B., Hothorn, T., Kneib, T., Schmid, M.: A framework for unbiased model selection based on boosting. J. Comput. Gr. Stat. **20**, 956–971 (2011)

Stat Comput (2018) 28:673–687 687

Hofner, B., Mayr, A., Fenske, N., Thomas, J., Schmid, M.: gamboostLSS: boosting methods for GAMLSS models (2017). http://CRAN.R-project.org/package=gamboostLSS. R package version 2.0-0

Hofner, B., Mayr, A., Robinzonov, N., Schmid, M.: Model-based boosting in R—A hands-on tutorial using the R package mboost. Comput. Stat. **29**, 3–35 (2014)

Hofner, B., Mayr, A., Schmid, M.: gamboostLSS: an R package for model building and variable selection in the GAMLSS framework. J. Stat. Softw. **74**(1), 1–31 (2016)

Hothorn, T., Buehlmann, P., Kneib, T., Schmid, M., Hofner, B.: Model-based boosting 2.0. J. Mach. Learn. Res. **11**, 2109–2113 (2010)

Hothorn, T., Buehlmann, P., Kneib, T., Schmid, T., Hofner, B.: mboost: model-based boosting (2017). http://CRAN.R-project.org/package=mboost. R package version 2.8-0

Hothorn, T., Müller, J., Schröder, B., Kneib, T., Brandl, R.: Decomposing environmental, spatial, and spatiotemporal components of species distributions. Ecol. Monogr. **81**, 329–347 (2011)

Huang, S.M.Y., Huang, J., Fang, K.: Gene network-based cancer prognosis analysis with sparse boosting. Genet. Res. **94**, 205–221 (2012)

Li, P.: Robust logitboost and adaptive base class (abc) logitboost (2012). arXiv preprint arXiv:1203.3491

Mayr, A., Binder, H., Gefeller, O., Schmid, M., et al.: The evolution of boosting algorithms. Methods Inf. Med. **53**(6), 419–427 (2014)

Mayr, A., Binder, H., Gefeller, O., Schmid, M., et al.: Extending statistical boosting. Methods Inf. Med. **53**(6), 428–435 (2014)

Mayr, A., Fenske, N., Hofner, B., Kneib, T., Schmid, M.: Generalized additive models for location, scale and shape for high-dimensional data—a flexible approach based on boosting. J. R. Stat. Soc. Ser. C Appl. Stat. **61**(3), 403–427 (2012)

Mayr, A., Hofner, B., Schmid, M.: Boosting the discriminatory power of sparse survival models via optimization of the concordance index and stability selection. BMC Bioinf. **17**(1), 288 (2016)

Mayr, A., Hofner, B., Schmid, M., et al.: The importance of knowing when to stop. Methods Inf. Med. **51**(2), 178–186 (2012)

Meinshausen, N., Bühlmann, P.: Stability selection. J. R. Stat. Soc. Ser. B (Stat. Methodol.) **72**(4), 417–473 (2010)

Messner, J.W., Mayr, G.J., Zeileis, A.: Nonhomogeneous boosting for predictor selection in ensemble postprocessing. Mon. Weather Rev. **145**(1), 137–147 (2017). doi:10.1175/MWR-D-16-0088.1

Mullahy, J.: Specification and testing of some modified count data models. J. Econom. **33**(3), 341–365 (1986)

Murtaugh, P.A.: Performance of several variable-selection methods applied to real ecological data. Ecol. Lett. **12**, 1061–1068 (2009)

Opelt, A., Fussenegger, M., Pinz, A., Auer, P.: Weak hypotheses and boosting for generic object detection and recognition. In: European Conference on Computer Vision, pp. 71–84. Springer (2004)

Osorio, J.D.G., Galiano, S.G.G.: Non-stationary analysis of dry spells in monsoon season of Senegal River Basin using data from regional climate models (RCMs). J. Hydrol. **450–451**, 82–92 (2012)

Rigby, R.A., Stasinopoulos, D.M.: Generalized additive models for location, scale and shape. J. R. Stat. Soc. Ser. C (Appl. Stat.) **54**(3), 507–554 (2005)

Rigby, R.A., Stasinopoulos, D.M., Akantziliotou, C.: Instructions on how to use the gamlss package in R (2008). http://www.gamlss.org/wp-content/uploads/2013/01/gamlss-manual.pdf

Schmid, M., Hothorn, T.: Boosting additive models using component-wise P-splines. Comput. Stat. Data Anal. **53**(2), 298–311 (2008)

Schmid, M., Potapov, S., Pfahlberg, A., Hothorn, T.: Estimation and regularization techniques for regression models with multidimensional prediction functions. Stat. Comput. **20**(2), 139–150 (2010)

Shah, R.D., Samworth, R.J.: Variable selection with error control: Another look at stability selection. J. R. Stat. Soc. Ser. B (Stat. Methodol.) **75**(1), 55–80 (2013)

Smith, A.D., Hofner, B., Osenkowski, J.E., Allison, T., Sadoti, G., McWilliams, S.R., Paton, P.W.C.: Spatiotemporal modelling of sea duck abundance: implications for marine spatial planning (2017). arXiv preprint arXiv:1705.00644

8. Automatic Exploration of Machine Learning Experiments on OpenML

Declaration of the specific contributions of the author The first version of the *Random Experimentation Bot*[*], that was used to gather the data was implemented by the author and later refined by Daniel Kühn and Philipp Probst. Daniel Kühn and Philipp Probst wrote the major parts of this publication. The author also contributed to the paper itself by refining and improving the description of the data and data collection process. Bernd Bischl helped improving the paper and the search space and algorithm selection for the data collection.

[*] www.github.com/ja-thomas/OMLbots

Automatic Exploration of Machine Learning Experiments on OpenML

Daniel Kühn*[a], Philipp Probst*[a], Janek Thomas[a], Bernd Bischl[a]

[a]*Ludwig-Maximilians-Universität München, Germany*

Abstract

Understanding the influence of hyperparameters on the performance of a machine learning algorithm is an important scientific topic in itself and can help to improve automatic hyperparameter tuning procedures. Unfortunately, experimental meta data for this purpose is still rare. This paper presents a large, free and open dataset addressing this problem, containing results on 38 OpenML data sets, six different machine learning algorithms and many different hyperparameter configurations. Results where generated by an automated random sampling strategy, termed the *OpenML Random Bot*. Each algorithm was cross-validated up to 20.000 times per dataset with different hyperparameters settings, resulting in a meta dataset of around 2.5 million experiments overall.

1. Introduction

When applying machine learning algorithms on real world datasets, users have to choose from a large selection of different algorithms with many of them offering a set of hyperparameters to control algorithmic performance. Although sometimes default values exist, there is no agreed upon principle for their definition (but see our recent work in in (Probst et al., 2018) for a potential approach). Automatic tuning of such parameters is a possible solution (Claesen and Moor, 2015), but comes with a considerable computational burden.

Meta-learning tries to decrease this cost (Feurer et al., 2015), by reusing information of previous runs of the algorithm on similar datasets, which obviously requires access to such prior empirical results. With this paper we provide a freely accessible meta dataset that contains around 2.5 million runs of six different machine learning algorithms on 38 classification datasets.

Large, freely available datasets like Imagenet (Deng et al., 2009) are important for the progress of machine learning, we hope to support developments in the area of meta-learning and benchmarking, meta-learning and hyperparameter tuning with our work here.

While similar meta-datasets have been created in the past, we were not able to access them by the links provided in their respective papers: Smith et al. (2014) provides a repository with Weka-based machine learning experiments on 72 data sets, 9 machine learning algorithms, 10 hyperparameter settings for each algorithm, and several meta-features of each data set. Reif (2012) created a meta-dataset based on machine learning experiments on 83 datasets, 6 classification algorithms, and 49 meta features.

In this paper, we describe our experimental setup, specify how our meta-dataset is created by running random machine learning experiments through the OpenML platform (Vanschoren et al., 2013) and explain how to access our results.

Email addresses: daniel.kuehn.87@gmail.com (Daniel Kühn*), philipp_probst@gmx.de (Philipp Probst*), janek.thomas@stat.uni-muenchen.de (Janek Thomas), bernd_bischl@gmx.net (Bernd Bischl)

2. Considered ML data sets, algorithms and hyperparameters

To create the meta dataset, six supervised machine learning algorithms are run on 38 classification tasks. For each algorithm the available hyperparameters are explored in a predefined range (see Table 1). Some of these hyperparameters are transformed by the function found in column *trafo* of Table 1 to allow non-uniform sampling, a usual procedure in tuning.

algorithm	hyperparameter	type	lower	upper	trafo
glmnet	alpha	numeric	0	1	-
	lambda	numeric	-10	10	2^x
rpart	cp	numeric	0	1	-
	maxdepth	integer	1	30	-
	minbucket	integer	1	60	-
	minsplit	integer	1	60	-
kknn	k	integer	1	30	-
svm	kernel	discrete	-	-	-
	cost	numeric	-10	10	2^x
	gamma	numeric	-10	10	2^x
	degree	integer	2	5	-
ranger	num.trees	integer	1	2000	-
	replace	logical	-	-	-
	sample.fraction	numeric	0	1	-
	mtry	numeric	0	1	$x \cdot p$
	respect.unordered.factors	logical	-	-	-
	min.node.size	numeric	0	1	n^x
xgboost	nrounds	integer	1	5000	-
	eta	numeric	-10	0	2^x
	subsample	numeric	0	1	-
	booster	discrete	-	-	-
	max_depth	integer	1	15	-
	min_child_weight	numeric	0	7	2^x
	colsample_bytree	numeric	0	1	-
	colsample_bylevel	numeric	0	1	-
	lambda	numeric	-10	10	2^x
	alpha	numeric	-10	10	2^x

Table 1: Hyperparameters of the algorithms. p refers to the number of variables and n to the number of observations. The used algorithms are `glmnet` (Friedman et al., 2010), `rpart` (Therneau and Atkinson, 2018), `kknn` (Schliep and Hechenbichler, 2016), `svm` (Meyer et al., 2017), `ranger` (Wright and Ziegler, 2017) and `xgboost` (Chen and Guestrin, 2016).

These algorithms are run on a subset of the OpenML100 benchmark suite (Bischl et al., 2017), which consists of 100 classification datasets, carefully curated from the thousands of datasets available on OpenML (Vanschoren et al., 2013). We only include datasets without missing data and with a binary outcome resulting in 38 datasets. The datasets and their respective characteristics can be found in Table 2.

Data_id	Task_id	Name	n	p	majPerc	numFeat	catFeat
3	3	kr-vs-kp	3196	37	0.52	0	37
31	31	credit-g	1000	21	0.70	7	14
37	37	diabetes	768	9	0.65	8	1
44	43	spambase	4601	58	0.61	57	1
50	49	tic-tac-toe	958	10	0.65	0	10
151	219	electricity	45312	9	0.58	7	2
312	3485	scene	2407	300	0.82	294	6
333	3492	monks-problems-1	556	7	0.50	0	7
334	3493	monks-problems-2	601	7	0.66	0	7
335	3494	monks-problems-3	554	7	0.52	0	7
1036	3889	sylva_agnostic	14395	217	0.94	216	1
1038	3891	gina_agnostic	3468	971	0.51	970	1
1043	3896	ada_agnostic	4562	49	0.75	48	1
1046	3899	mozilla4	15545	6	0.67	5	1
1049	3902	pc4	1458	38	0.88	37	1
1050	3903	pc3	1563	38	0.90	37	1
1063	3913	kc2	522	22	0.80	21	1
1067	3917	kc1	2109	22	0.85	21	1
1068	3918	pc1	1109	22	0.93	21	1
1120	3954	MagicTelescope	19020	12	0.65	11	1
1461	14965	bank-marketing	45211	17	0.88	7	10
1462	10093	banknote-authentication	1372	5	0.56	4	1
1464	10101	blood-transfusion-service-center	748	5	0.76	4	1
1467	9980	climate-model-simulation-crashes	540	21	0.91	20	1
1471	9983	eeg-eye-state	14980	15	0.55	14	1
1479	9970	hill-valley	1212	101	0.50	100	1
1480	9971	ilpd	583	11	0.71	9	2
1485	9976	madelon	2600	501	0.50	500	1
1486	9977	nomao	34465	119	0.71	89	30
1487	9978	ozone-level-8hr	2534	73	0.94	72	1
1489	9952	phoneme	5404	6	0.71	5	1
1494	9957	qsar-biodeg	1055	42	0.66	41	1
1504	9967	steel-plates-fault	1941	34	0.65	33	1
1510	9946	wdbc	569	31	0.63	30	1
1570	9914	wilt	4839	6	0.95	5	1
4134	14966	Bioresponse	3751	1777	0.54	1776	1
4534	34537	PhishingWebsites	11055	31	0.56	0	31

Table 2: Included datasets and respective characteristics. n are the number of observations, p the number of features, *maj.class* the percentage of observations in the largest class, *numFeat* the number of numeric features and *catFeat* the number of categorical features.

3. Random Experimentation Bot

To conduct a large number of experiments a bot was implemented to automatically plan and execute runs, following the paradigm of random search. The bot iteratively executes these steps:

1. Randomly sample a task T (with an associated data set) from Table 2.
2. Randomly sample one ML algorithm A.
3. Randomly sample a hyperparameter setting θ of algorithm A, uniformly from the ranges specified in Table 1, then transform, if a transformation function is given.
4. Obtain task T (and dataset) from OpenML and store it locally.
5. Evaluate algorithm A with configuration θ on task T, with associated 10-fold cross-validation from OpenML.
6. Upload run results to OpenML, including hyperparameter configuration and time measurements.
7. OpenML now calculates various performance metrics for the uploaded cross-validated predictions.
8. The OpenML-ID of the bot (2702) and the tag `mlrRandomBot` is used for identification.

A clear advantage of random sampling is that all bot runs are completely independent of each other, making all experiments embarrassingly parallel. Furthermore, more experiments can easily and conveniently added later on, without introducing any kind of bias into the sampling method.

The bot is developed open source in R and can be found on GitHub[1]. The bot is based on the R packages `mlr` (Bischl et al., 2016) and `OpenML` (Casalicchio et al., 2017) and written in modular form such that it can be extended with new sampling strategies for hyperparameters, algorithms and datasets in the future. Parallelization was performed with R package `batchtools` (Lang et al., 2017).

After more than 6 million benchmark experiments the results of the bot are downloaded from OpenML. For each of the algorithms 500000 experiments are used to obtain the final dataset. The experiments are chosen by the following procedure: For each algorithm, a threshold B is set (see below) and, if the number of results for a dataset exceeds B, we draw randomly B of the results obtained for this algorithm and this dataset. The threshold value B is chosen for each algorithm separately to exactly obtain in total 500000 results for each algorithm.

For `kknn` we only execute 30 experiments per dataset because this number of experiments is high enough to cover the hyperparameter space (that only consists of the parameter k for $k \in \{1, ..., 30\}$) appropriately, resulting in 1140 experiments. All in all this results in around 2.5 million experiments.

The distribution of the runs on the datasets and algorithms is displayed in Table 3.

Data_id	Task_id	glmnet	rpart	kknn	svm	ranger	xgboost	Total
3	3	15547	14633	30	19644	15139	16867	81860
31	31	15547	14633	30	19644	15139	16867	81860
37	37	15546	14633	30	15985	15139	16866	78199
44	43	15547	14633	30	19644	15139	16867	81860
50	49	15547	14633	30	19644	15139	16866	81859
151	219	15547	14632	30	2384	12517	16866	61976
312	3485	6613	13455	30	18740	12985	15886	67709
333	3492	15546	14632	30	19644	15139	16867	81858
334	3493	15547	14633	30	19644	14492	16867	81213
335	3494	15547	14633	30	15123	15139	10002	70474
1036	3889	14937	14633	30	2338	7397	2581	41916
1038	3891	15547	5151	30	5716	4827	1370	32641
1043	3896	6466	14633	30	10121	3788	16867	51905
1046	3899	15547	14633	30	5422	8842	11812	56286
1049	3902	7423	14632	30	12064	15139	4453	53741
1050	3903	15547	14633	30	19644	11357	13758	74969
1063	3913	15547	14633	30	19644	7914	16866	74634
1067	3917	15546	14632	30	10229	7386	16866	64689
1068	3918	15546	14633	30	13893	8173	16866	69141
1120	3954	15531	7477	30	3908	9760	8143	44849
1461	14965	6970	14073	30	2678	14323	2215	40289
1462	10093	8955	14633	30	6320	15139	16867	61944
1464	10101	15547	14632	30	19644	15139	16867	81859
1467	9980	15547	14633	30	4441	15139	16866	66656
1471	9983	15547	14633	30	9725	13523	16866	70324
1479	9970	15546	14633	30	19644	15140	16867	81860
1480	9971	15024	14633	30	19644	15139	16254	80724
1485	9976	8247	10923	30	10334	15139	9237	53910
1486	9977	3866	11389	30	1490	15139	5813	37727
1487	9978	15547	6005	30	19644	15139	11194	67559
1489	9952	15547	14633	30	17298	15139	16867	79514
1494	9957	15547	14632	30	19644	15139	16867	81859
1504	9967	15547	14633	30	19644	15140	16867	81861
1510	9946	15547	14633	30	19644	15139	16867	81860
1570	9914	15546	14632	30	19644	15139	16867	81858
4134	14966	1493	3947	30	560	14516	2222	22768
4534	34537	2801	3231	30	2476	15139	947	24624
Total	257661	486995	485368	1110	485549	484860	486953	2430835

Table 3: Number of experiments for each combination of dataset and algorithm.

[1] `https://github.com/ja-thomas/OMLbots`

4. Access to the results

The results of the benchmark can be accessed in different ways:

- The easiest way to access them is to go to the figshare repository (Kühn et al., 2018) and to download the `.csv` files. For each algorithm there is a csv file that contains a row for each algorithm run with the columns `Data_id`, the hyperparameter settings, the performance measures (auc, accuracy and brier score), the runtime, the scimark reference runtime and some characteristics of the dataset such as the number of features or the number of observations.
- Alternatively the code for the extraction of the data from the nightly database snapshot of OpenML can be found here: `https://github.com/ja-thomas/OMLbots/blob/master/snapshot_database/database_extraction.R`. With this script all results that were created by the random bot (OpenML-ID 2702) are downloaded and the final dataset is created. (Warning: As the OpenML database is updated daily, changes can occur.)

5. Discussion and potential usage of the results

The presented data can be used to study the effect and influence of hyperparameter setting on performance in various ways. Possible applications are:

- Obtaining defaults for ML algorithm that work well across many datasets (Probst et al., 2018);
- Measuring the importance of hyperparameters, to investigate which should be tuned (see van Rijn and Hutter, 2017; Probst et al., 2018);
- Obtaining ranges or priors of tuning parameters to focus on important regions of the search space (see van Rijn and Hutter, 2017; Probst et al., 2018);
- Meta-Learning;
- Investigating, debugging and improving the robustness of algorithms.

Possible weaknesses of the approach, which we would like to address in the future, are:

- For each ML algorithm, a set of considered hyperparameters and their initial ranges has to be provided. It would be much more convenient if the bot could handle the set of all technical hyperparameters, with infinite ranges.
- Smarter, sequential sampling might be required to scale to high-dimensional hyperparameter spaces. But note that we not only care about optimal configurations but much rather would like to learn as much as possible about the considered parameter space, including areas of bad performance. So simply switching to Bayesian optimization or related search techniques might not be appropriate.

References

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in R. *Journal of Machine Learning Research*, 17 (170):1–5, 2016.

B. Bischl, G. Casalicchio, M. Feurer, F. Hutter, M. Lang, R. G. Mantovani, J. N. van Rijn, and J. Vanschoren. OpenML benchmarking suites and the OpenML100. *ArXiv preprint arXiv:1708.03731*, Aug. 2017. URL `https://arxiv.org/abs/1708.03731`.

G. Casalicchio, J. Bossek, M. Lang, D. Kirchhoff, P. Kerschke, B. Hofner, H. Seibold, J. Vanschoren, and B. Bischl. OpenML: An R package to connect to the machine learning platform OpenML. *Computational Statistics*, 32(3):1–15, 2017.

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*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

M. Claesen and B. D. Moor. Hyperparameter search in machine learning. *MIC 2015: The XI Metaheuristics International Conference*, 2015.

J. Deng, W. Dong, R. Socher, L. jia Li, K. Li, and L. Fei-fei. Imagenet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

M. Feurer, J. T. Springenberg, and F. Hutter. Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1128–1135. AAAI Press, 2015.

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

D. Kühn, P. Probst, J. Thomas, and B. Bischl. OpenML R bot benchmark data (final subset). 2018. URL `https://figshare.com/articles/OpenML_R_Bot_Benchmark_Data_final_subset_/5882230`.

M. Lang, B. Bischl, and D. Surmann. batchtools: Tools for R to work on batch systems. *The Journal of Open Source Software*, 2(10), 2017.

D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. L. h. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2017. URL `https://CRAN.R-project.org/package=e1071`. R package version 1.6-8.

P. Probst, B. Bischl, and A.-L. Boulesteix. Tunability: Importance of hyperparameters of machine learning algorithms. *ArXiv preprint arXiv:1802.09596*, 2018. URL `https://arxiv.org/abs/1802.09596`.

M. Reif. A comprehensive dataset for evaluating approaches of various meta-learning tasks. In *ICPRAM*, 2012.

K. Schliep and K. Hechenbichler. *kknn: Weighted k-Nearest Neighbors*, 2016. URL `https://CRAN.R-project.org/package=kknn`. R package version 1.3.1.

M. R. Smith, A. White, C. Giraud-Carrier, and T. Martinez. An easy to use repository for comparing and improving machine learning algorithm usage. In *Meta-Learning and algorithm selection workshop at ECAI 2014*, page 41, 2014.

T. Therneau and B. Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2018. URL `https://CRAN.R-project.org/package=rpart`. R package version 4.1-12.

J. N. van Rijn and F. Hutter. Hyperparameter importance across datasets. *ArXiv preprint arXiv:1710.04725*, 2017. URL `https://arxiv.org/abs/1710.04725`.

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked Science in Machine Learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

M. N. Wright and A. Ziegler. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017.

9. Automatic Gradient Boosting

Declaration of the specific contributions of the author The implementation of `autoxgboost` as well as the paper itself was mainly written by the author. Stefan Coors conducted the initial benchmark study and helped defining parts of the algorithm, mainly the feature encoding and threshold optimization as part of his master thesis. Bernd Bischl helped designing important aspect of the proposed AutoML framework and helped defining the benchmark study.

ICML 2018 AutoML Workshop

Automatic Gradient Boosting

Janek Thomas JANEK.THOMAS@STAT.UNI-MUENCHEN.DE
Stefan Coors STEFAN.COORS@CAMPUS.LMU.DE
Bernd Bischl BERND.BISCHL@STAT.UNI-MUENCHEN.DE
Department of Statistics, LMU, Ludwigstrasse 33, D80539 Munich

Abstract

Automatic machine learning performs predictive modeling with high performing machine learning tools without human interference. This is achieved by making machine learning applications parameter-free, i.e. only a dataset is provided while the complete model selection and model building process is handled internally through (often meta) optimization. Projects like Auto-WEKA and auto-sklearn aim to solve the Combined Algorithm Selection and Hyperparameter optimization (CASH) problem resulting in huge configuration spaces. However, for most real-world applications, the optimization over only a few different key learning algorithms can not only be sufficient, but also potentially beneficial. The latter becomes apparent when one considers that models have to be validated, explained, deployed and maintained. Here, less complex model are often preferred, for validation or efficiency reasons, or even a strict requirement. Automatic gradient boosting simplifies this idea one step further, using only gradient boosting as a single learning algorithm in combination with model-based hyperparameter tuning, threshold optimization and encoding of categorical features. We introduce this general framework as well as a concrete implementation called autoxgboost. It is compared to current AutoML projects on 16 datasets and despite its simplicity is able to achieve comparable results on about half of the datasets as well as performing best on two.

Keywords: AutoML, Gradient Boosting, Bayesian Optimization, Machine Learning

1. Introduction

Machine Learning, *Predictive Modeling* and *Artificial Intelligence* are ongoing topics in research as well as in industrial applications. While data are gathered everywhere nowadays, many potential insights are often not fully achieved since data science and ML experts are still a rare commodity. While many stages of a data analysis project still need to be done manually by human data scientists, model search and optimization can be done automatically. Automatic machine learning (AutoML) simplifies the workload by making decisions for common predictive modeling tasks like regression or classification. We distinguish between *single-learner* AutoML methods which aim to make single algorithms parameter-free and more general approaches, which combine several learning algorithms into one optimization problem. These *multi-learner* methods solve the *Combined Algorithm Selection and Hyperparameter optimization (CASH)* problem (Thornton et al. (2013)). Modern approaches that include pre- and postpressing methods are referred to as *machine learning pipeline configuration*.

There is a growing number of open source approaches for automating machine learning available for non-professionals. As one of the first frameworks, Auto-WEKA (Thornton et al. (2013)) introduced a system for automatically choosing from a broad variety of learning algorithms implemented in the open source software *WEKA* (Hall et al. (2009)). Hereby, Auto-WEKA simultaneously tunes hyperparameters over all learning algorithms model using the Bayesian optimization framework SMAC (Hutter et al. (2011)). Similar to Auto-WEKA is auto-sklearn (Feurer et al. (2015)), which is based on the scikit-learn toolkit for python and includes all of its learners as well as available preprocessing operations. It stacks multiple models to achieve high predictive performance. Another python-based AutoML tool is called *Tree-based Pipeline Optimization Tool (TPOT)* by Olson et al. (2016) and uses genetic programming instead of Bayesian optimization to tune over a similar space as auto-sklearn.

Only few *single-learner* AutoML methods exist. A lot of services, for example Google's *Cloud AutoML*, focus on specialized application domains like image recognition using deep neural networks. For general machine learning tasks, Probst et al. (2018) introduced the tuneRanger software, which automatically tunes a random forest. Another algorithm approach is called *Parameter-free STOchastic Learning (PiSTOL)* (Orabona (2014)) and directly tries to optimize the generalization performance of a learning algorithm, in a stochastic approximation way.

Our proposed approach reduces the AutoML framework to the construction of an optimal gradient boosting model (Friedman (2001)), which is a strong predictive algorithm, as long as its hyperparameters are adequately tuned. Besides tuning the hyperparameters via Bayesian optimization, categorical feature transformation is performed as a preprocessing step. Moreover, for classification tasks, thresholds are optimized repeatedly. By focusing on a single learning algorithm, hyperparameters can be optimized much more thoroughly and the resulting model can be analyzed and deployed much easier.

2. Method

This section introduces the structure of the automatic gradient boosting framework. The general workflow of the approach can be seen in Figure 1. Automatic gradient boosting uses gradient boosting with trees (GBT) as its only learning algorithm. GBT is popular due to its strong predictive performance and robustness. A large number of machine learning competitions were won by these algorithms, see Chen and Guestrin (2016) for an overview. Furthermore it possesses multiple highly desirable properties for an AutoML system: It is insensitive to outliers, as the trees used in gradient boosting are invariant to monotone transformations of the data, which makes scaling the data obsolete; GBT implementations are usually able to handle missing values in the data directly by learning default split directions for missing values (Chen and Guestrin (2016)); In addition, they are capable of handling high dimensional feature spaces, as features are evaluated separately for each split in a tree, which can be parallelized and does not result in a harder optimization problem for more features. One last important aspect to consider is that boosting can be easily adapted to tasks like ranking (Li et al. (2008)) or survival analysis (Chen et al. (2013)). Modern GBT frameworks like xgboost (Chen and Guestrin (2016)) or lightgbm (Ke et al. (2017)) are highly configurable with a large number of hyperparameters for regularization and

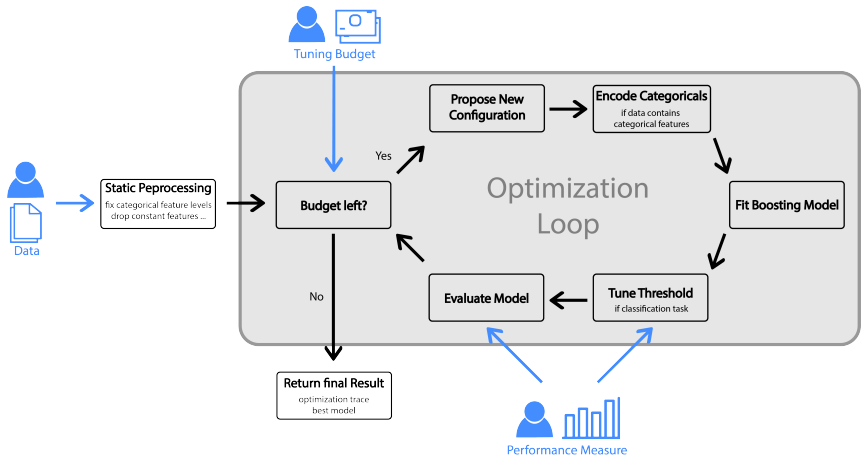

Figure 1: Workflow of the automatic gradient boosting approach. Blue lines indicate input by human.

optimization. With the recent addition of dropout boosting (Rashmi and Gilad-Bachrach (2015)) to these frameworks, it is possible to let the boosting algorithms behave similar, or even identical to random forests by setting hyperparameters accordingly.

The large number of hyperparameters makes tuning for GBT a necessity. Different methods like grid or random search can be used for simple hyperparameter optimization problems, but to achieve more efficient optimization, adaptive strategies should be employed. *Sequential model-based optimization (SMBO)*, also known as *Bayesian optimization* is one of the state-of-the-art adaptive hyperparameter optimization strategies (Snoek et al. (2012)). Depending on the difficulty of the hyperparameter space in terms of categorical and dependent hyperparameters, different surrogate models are used. In Table 1 two different possible hyperparameter spaces are proposed, a fully numeric one (denoted Simple=Y), which can be optimized with Gaussian process surrogate models, and a more complex one (denoted Simple=N), which can be optimized with a random forest surrogate. Arguably the most important hyperparameter in GBT is the number of boosting iterations, which is efficiently found by early-stopping, i.e., measuring the performance on validation data after each iteration. An advantage of combining early stopping with SMBO is that the validation error can be directly returned to the optimizer without the need of an additional holdout set or resampling. No necessity for internal resampling and the parallel implementation of GBT algorithms allow to mitigate the disadvantage of the sequential nature of SMBO without using parallel SMBO variants (see Bischl et al. (2014) for an overview), such that parallel system architectures can be fully utilized.

Most boosting implementations cannot natively handle categorical variables and it is necessary to transform such features. The simplest possibility is to encode these features into integers, with the drawback that the optimal order is unknown and a random one is used. The concept of dummy encoding is to create one separate feature for each level,

i.e., a feature x with levels a, b, c is dummy encoded into binary features x_a^*, x_b^* and x_c^*. No information is lost by this encoding but it can be infeasible for high cardinality features with a high number of unique feature levels. A third method of transforming categorical features is *impact encoding* (Micci-Barreca (2001)). Features are encoded by replacing categories with aggregated values of the target in the respective group, e.g., $\bar{y}|_{x=a}$ for regression or $P(y|x = a)$ for classification. We evaluate different combinations of these encodings, mainly based on a threshold, e.g., features with less than k levels are dummy encoded while integer or impact encoding is done for the remaining categorical features. It is also possible to tune this threshold k together with the GBT hyperparameters. For datasets with few categorical features the encoding can also be learned separately for each feature.

Depending on the overall performance metric that should be optimized, it can be difficult to find the best loss function for GBT since not every performance metric can be directly plugged in as loss functions. For binary- and multiclass classification it is often useful to optimize classification thresholds of each class directly with regard to the used performance metric. The threshold is optimized for each iteration of the model-based optimization on the validation set that was already used for early stopping. The resulting internal performance value is hence biased, but since the tuning error of the optimizer is biased anyways, we let this slide, especially since reducing training data size or extra resampling is much less efficient. This design decision should be investigated in more details in future studies, though.

Combining all of the above components, we achieve a fast, scalable and robust AutoML solution that can handle categorical parameters (even with many levels), outliers and missing data, while having a much smaller configuration space compared to existing solutions.

3. Implementation

This section introduces an implementation of the described automatic gradient boosting framework. In general, the introduced framework could use a large number of available boosting library (e.g., xgboost or lightgbm) as well as different SMBO libraries (e.g., SMAC, Spearmint or mlrMBO). We decided to implement it in R using xgboost as a GBT implementation, mlrMBO (Bischl et al. (2017)) for SMBO and mlr (Bischl et al. (2016)) as a general machine learning framework as well as for threshold optimization. For threshold optimization a multi-start linesearch is used for binary classification and for multiclass classification *Generalized Simulated Annealing (GSA)* (Tsallis and Stariolo (1996)) is applied. The software is available via Github[1] and is currently able to handle binary and multiclass classification as well as regression. It can be used in a standalone version or within the mlr framework as a learner. Other than the data itself no further information has to be passed to autoxgboost, but it may be useful to define the performance metric (otherwise a default will be used depending on the type of the data) and the maximum runtime. The result is a reusable machine learning pipeline based on the library mlrCPO[2] that can be deployed or saved for later use. Currently two different hyperparameter spaces are predefined (see Table 1) and the simpler one (where Simple=Y) is used by default.

1. https://github.com/ja-thomas/autoxgboost
2. https://github.com/mlr-org/mlrCPO

4. Benchmark

We compare the performance of autoxgboost to the AutoML solutions Auto-WEKA and auto-sklearn. In order to ensure comparability, we evaluate autoxgboost on a subset[3] of the datasets Auto-WEKA and auto-sklearn used in their respective publications. This includes identical training- and test-data splits and the same performance measure. The chosen datasets are very different regarding the number of numeric and factor features, as well as the number of target class levels and the train and test dataset sizes. Hence, the datasets chosen by Thornton et al. (2013) serve as an adequate heterogeneous base for an initial performance evaluation in different situations. Moreover, like in the paper of Thornton et al. (2013), 25 runs were performed.
The parameter settings of autoxgboost were mostly left at their default values discussed in the previous section. However, at most 160 tuning iterations were allowed with a maximum runtime of 10 hours. The benchmark was run on Intel Xeon E5-2697 v3 processors with 28 cores and 64gb RAM. The hyperparameter ranges corresponded to the ones from Table 1 (simple).

For evaluation, 100000 bootstrap samples of size 4 were drawn from all 25 runs to simulate 4 parallel runs. Finally, the median of those 100000 mean misclassification error values is returned and presented in Table 2. The bold numbers in each row indicates the best performing algorithm for the specific dataset. We added a simple majority class baseline

3. The subset was selected to reduce computational demand. It was not cherry picked or altered in any way to improve results. A larger benchmark on more datasets is planned. An overview of the datasets can be found at https://github.com/ja-thomas/autoxgboost

Name	**Range**	**Dependency**	**$\log_2$ scale**	**Simple**
eta	$[0.01, 0.2]$		N	Y
gamma	$[-7, 6]$		Y	Y
max_depth	$\{3, 4, \ldots, 20\}$		N	Y
colsample_bytree	$[0.5, 1]$		N	Y
colsample_bylevel	$[0.5, 1]$		N	Y
lambda	$[-10, 10]$		Y	Y
alpha	$[-10, 10]$		Y	Y
subsample	$[0.5, 1]$		N	Y
booster	gbtree, gblinear, dart		N	N
sample_type	uniform weighted	dart	N	N
normalize_type	tree, forest	dart	N	N
rate_drop	$[0, 1]$	dart	N	N
skip_drop	$[0, 1]$	dart	N	N
one_drop	TRUE, FALSE	dart	N	N
grow_policy	depthwise, lossguide		N	N
max_leaves	$\{0, 1, \ldots, 8\}$	lossguide	Y	N
max_bin,	$\{2, 3, \ldots, 9\}$		Y	N

Table 1: Proposed hyperparameter spaces to tune over in autoxgboost. The first 8 parameters are defined as the simple space (default).

Dataset	baseline	autoxgboost	Auto-WEKA	auto-sklearn
Dexter	52.78	12.22	7.22	**5.56**
GermanCredit	32.67	27.67*	28.33	**27.00**
Dorothea	6.09	**5.22**	6.38	5.51
Yeast	68.99	**38.88**	40.45	40.67
Amazon	99.33	26.22	37.56	**16.00**
Secom	**7.87**	**7.87**	**7.87**	**7.87**
Semeion	92.45	8.38	**5.03**	5.24
Car	29,15	1.16	0.58	**0.39**
Madelon	50.26	16.54	21.15	**12.44**
KR-vs-KP	48.96	1.67	**0.31**	0.42
Abalone	84.04	73.75*	**73.02**	73.50
Wine Quality	55.68	**33.70**	**33.70**	33.76
Waveform	68.80	15.40*	**14.40**	14.93
Gisette	50.71	2.48	2.24	**1.62**
Convex	50.00	22.74	22.05	**17.53**
Rot. MNIST + BI	88.88	47.09*	55.84	**46.92**

Table 2: Benchmark results are median percent error across 100 000 bootstrap samples (out of 25 runs) simulating 4 parallel runs. Bold numbers indicate best performing algorithms. Stars indicate a relative difference of less than 5% to auto-sklearn.

as an indicator that all implementations work as they should, i.e., they should significantly outperform this baseline. As we can see easily in Table 2, only for the dataset *Secom*, the baseline achieves the same performance as the AutoML frameworks. On 9 of the 16 datasets, auto-sklearn provides the best results. So does Auto-WEKA on four and autoxgboost on two datasets. autoxgboost and Auto-WEKA slightly perform better than auto-sklearn on the *Wine Quality* dataset.

5. Conclusion

The benchmark results of Section 4 showed that autoxgboost was outperformed on the larger number of datasets by auto-sklearn, but was able to achieve competitive results on some datasets, providing state-of-the-art performance with only a single learning algorithm instead of using a whole library of possibly ensembled algorithms. This is not too surprising as the tuning space is much smaller and on some of the datasets very different learning algorithms might have an edge. Obviously, this is only a small initial benchmark that is not necessarily representative. We plan to evaluate on a larger set of OpenML (Vanschoren et al. (2014)) datasets in the future. One clear advantage of this approach is that the resulting models are boosting models, which can be deployed more easily and allow some form of interpretability for example via feature importance and individualized feature attribution (Lundberg and Lee (2017)). Our AutoML implementation autoxgboost is still in an early state and some of the design decisions are not final and will be evaluated and optimized in the future. Furthermore, we plan to extend the automatic gradient boosting framework to optimize for simultaneously sparse and well performing models using multiobjective SMBO strategies by Horn and Bischl (2016).

References

Bernd Bischl, Simon Wessing, Nadja Bauer, Klaus Friedrichs, and Claus Weihs. MOI-MBO: multiobjective infill for parallel model-based optimization. In *International Conference on Learning and Intelligent Optimization*, pages 173–186. Springer, 2014.

Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: Machine learning in R. *Journal of Machine Learning Research*, 17(170):1–5, 2016.

Bernd Bischl, Jakob Richter, Jakob Bossek, Daniel Horn, Janek Thomas, and Michel Lang. *mlrMBO: A Modular Framework for Model-Based Optimization of Expensive Black-Box Functions*, 2017.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2.

Yifei Chen, Zhenyu Jia, Dan Mercola, and Xiaohui Xie. A gradient boosting algorithm for survival analysis via direct optimization of concordance index. *Computational and mathematical methods in medicine*, 2013, 2013.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 10 2001.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11 (1):10–18, November 2009. ISSN 1931-0145.

Daniel Horn and Bernd Bischl. Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pages 1–8. IEEE, 2016.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. *Sequential Model-Based Optimization for General Algorithm Configuration*, pages 507–523. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-25566-3.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3149–3157. Curran Associates, Inc., 2017.

Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*, pages 897–904, 2008.

Scott M Lundberg and Su-In Lee. Consistent feature attribution for tree ensembles. *arXiv preprint arXiv:1706.06060*, 2017.

Daniele Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *SIGKDD Explor. Newsl.*, 3(1):27–32, July 2001. ISSN 1931-0145.

Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. Automating biomedical data science through tree-based pipeline optimization. In Giovanni Squillero and Paolo Burelli, editors, *Applications of Evolutionary Computation*, pages 123–137, Cham, 2016. Springer International Publishing. ISBN 978-3-319-31204-0.

Francesco Orabona. Simultaneous model selection and optimization through parameter-free stochastic learning. In *Advances in Neural Information Processing Systems*, pages 1116–1124, 2014.

Philipp Probst, Marvin Wright, and Anne-Laure Boulesteix. Hyperparameters and tuning strategies for random forest. *arXiv preprint arXiv:1804.03515*, 2018.

KV Rashmi and Ran Gilad-Bachrach. Dart: Dropouts meet multiple additive regression trees. In *International Conference on Artificial Intelligence and Statistics*, pages 489–497, 2015.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD-2013*, pages 847–855, 2013.

Constantino Tsallis and Daniel A. Stariolo. Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications*, 233(1):395 – 406, 1996. ISSN 0378-4371.

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*, pages 897–904, 2008.

Scott M Lundberg and Su-In Lee. Consistent feature attribution for tree ensembles. *arXiv preprint arXiv:1706.06060*, 2017.

Daniele Micci-Barreca. A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *SIGKDD Explor. Newsl.*, 3(1):27–32, July 2001. ISSN 1931-0145.

Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. Automating biomedical data science through tree-based pipeline optimization. In Giovanni Squillero and Paolo Burelli, editors, *Applications of Evolutionary Computation*, pages 123–137, Cham, 2016. Springer International Publishing. ISBN 978-3-319-31204-0.

Francesco Orabona. Simultaneous model selection and optimization through parameter-free stochastic learning. In *Advances in Neural Information Processing Systems*, pages 1116–1124, 2014.

Philipp Probst, Marvin Wright, and Anne-Laure Boulesteix. Hyperparameters and tuning strategies for random forest. *arXiv preprint arXiv:1804.03515*, 2018.

KV Rashmi and Ran Gilad-Bachrach. Dart: Dropouts meet multiple additive regression trees. In *International Conference on Artificial Intelligence and Statistics*, pages 489–497, 2015.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD-2013*, pages 847–855, 2013.

Constantino Tsallis and Daniel A. Stariolo. Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications*, 233(1):395 – 406, 1996. ISSN 0378-4371.

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

10. compboost: Modular Framework for Component-Wise Boosting

Declaration of the specific contributions of the author The implementation of `compboost` was mainly done by Daniel Schalk as part of his master thesis. The author contributed the software design and structure. He wrote the initial version of the R interface to the software. The paper was mainly written by Daniel Schalk and improved and iterated by all authors.

The Journal of Open Source Software

compboost: Modular Framework for Component-Wise Boosting

Daniel Schalk[1], **Janek Thomas**[1], **and Bernd Bischl**[1]

1 Department of Statistics, LMU Munich

DOI: 10.21105/joss.00894

Software

- Review
- Repository
- Archive

Submitted: 14 August 2018
Published: 20 August 2018

Summary

In high-dimensional prediction problems, especially in the $p \geq n$ situation, feature selection is an essential tool. A fundamental method for problems of this type is component-wise gradient boosting, which automatically selects from a pool of base learners – e.g. simple linear effects or component-wise smoothing splines (Schmid and Hothorn 2008) – and produces a sparse additive statistical model. Boosting these kinds of models maintains interpretability and enables unbiased model selection in high-dimensional feature spaces (Hofner et al. 2012).

The `R` (Team 2016) package `compboost`, which is actively developed on GitHub (https://github.com/schalkdaniel/compboost), implements component-wise boosting in `C++` using `Rcpp` (Eddelbuettel 2013) and `Armadillo` (Sanderson and Curtin 2016) to achieve efficient runtime behavior and full memory control. It provides a modular object-oriented system which can be extended with new base-learners, loss functions, optimization strategies, and stopping criteria, either in `R` for convenient prototyping or directly in `C++` for optimized speed. The latter extensions can be added at runtime, without recompiling the whole framework. This allows researchers to easily implement more specialized base-learners, e.g., for spatial or random effects, used in their respective research area.

Visualization of selected effects, efficient adjustment of the number of iterations, and traces of selected base-learners and losses to obtain information about feature importance are supported.

Compared to the reference implementation for component-wise gradient boosting in `R`, `mboost` (Hothorn et al. 2017), `compboost` is optimized for larger datasets and easier to extend, even though it currently lacks some of the large functionality `mboost` provides. A detailed benchmark against `mboost` can be viewed on the project homepage and on GitHub.

The modular design of `compboost` allows extension to more complicated settings like functional data or survival analysis. Further work on the package should include parallelized boosting, better feature selection, faster optimization techniques such as momentum and adaptive learning rates, as well as better overfitting control.

References

Eddelbuettel, Dirk. 2013. *Seamless R and C++ Integration with Rcpp*. Springer. https://doi.org/10.1007/978-1-4614-6868-4.

Hofner, Benjamin, Torsten Hothorn, Thomas Kneib, and Matthias Schmid. 2012. "A Framework for Unbiased Model Selection Based on Boosting." *Journal of Computational*

The Journal of Open Source Software

and Graphical Statistics 20 (4). Taylor & Francis:956–71. https://doi.org/10.1198/jcgs.2011.09220.

Hothorn, Torsten, Peter Buehlmann, Thomas Kneib, Matthias Schmid, and Benjamin Hofner. 2017. *mboost: Model-Based Boosting*. https://CRAN.R-project.org/package=mboost.

Sanderson, Conrad, and Ryan Curtin. 2016. "Armadillo: A Template-Based C++ Library for Linear Algebra." *Journal of Open Source Software* 1 (2). Journal of Open Source Software:26. https://doi.org/10.21105/joss.00026.

Schmid, Matthias, and Torsten Hothorn. 2008. "Boosting Additive Models Using Component-Wise P-Splines." *Computational Statistics & Data Analysis* 53 (2). Elsevier:298–311.

Team, R Core. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Further References

Aha, D. W. and R. L. Bankert (1996). A comparative evaluation of sequential feature selection algorithms. In *Learning from data*, pp. 199–206. Springer.

Altmann, A., L. Toloşi, O. Sander, and T. Lengauer (2010). Permutation importance: a corrected feature importance measure. *Bioinformatics 26*(10), 1340–1347.

Barber, R. F., E. J. Candès, et al. (2015). Controlling the false discovery rate via knockoffs. *The Annals of Statistics 43*(5), 2055–2085.

Bartz-Beielstein, T., M. Friese, M. Zaefferer, B. Naujoks, O. Flasch, W. Konen, and P. Koch (2011). Noisy optimization with sequential parameter optimization and optimal computational budget allocation. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pp. 119–120. ACM.

Bartz-Beielstein, T., C. W. Lasarczyk, and M. Preuß (2005). Sequential parameter optimization. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, Volume 1, pp. 773–780. IEEE.

Bell, D. A. and H. Wang (2000). A formalism for relevance and its application in feature subset selection. *Machine learning 41*(2), 175–195.

Bergstra, J., D. Yamins, and D. D. Cox (2013). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, pp. 13–20. Citeseer.

Bergstra, J. S., R. Bardenet, Y. Bengio, and B. Kégl (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pp. 2546–2554.

Bischl, B., S. Wessing, N. Bauer, K. Friedrichs, and C. Weihs (2014). Moi-mbo: multiobjective infill for parallel model-based optimization. In *International Conference on Learning and Intelligent Optimization*, pp. 173–186. Springer.

Breiman, L. (1984). *Classification and regression trees*. Routledge.

Breiman, L. (1996). Bagging predictors. *Machine learning 24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine learning 45*(1), 5–32.

Bühlmann, P., T. Hothorn, et al. (2007). Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science 22*(4), 477–505.

Burges, C. J. (2010). From ranknet to lambdarank to lambdamart: An overview. *Learning 11*(23-581), 81.

Candes, E., Y. Fan, L. Janson, and J. Lv (2018). Panning for gold:'model-x'knockoffs for high dimensional controlled variable selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 80*(3), 551–577.

Caruana, R., A. Niculescu-Mizil, G. Crew, and A. Ksikes (2004). Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 18. ACM.

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

Chen, Y., Z. Jia, D. Mercola, and X. Xie (2013). A gradient boosting algorithm for survival analysis via direct optimization of concordance index. *Computational and mathematical methods in medicine 2013*.

Cheng, S. Y., Y. Chen, D. Khosla, and K. Kim (2011). Optimal multiclass classifier threshold estimation with particle swarm optimization for visual object recognition. In *International Symposium on Visual Computing*, pp. 536–544. Springer.

Clarke, R., H. W. Ressom, A. Wang, J. Xuan, M. C. Liu, E. A. Gehan, and Y. Wang (2008). The properties of high-dimensional data spaces: implications for exploring gene and protein expression data. *Nature reviews cancer 8*(1), 37.

Eddelbuettel, D. and R. François (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software 40*(8), 1–18.

Eggensperger, K., F. Hutter, H. H. Hoos, and K. Leyton-Brown (2015). Efficient benchmarking of hyperparameter optimizers via surrogates. In *AAAI*, pp. 1114–1120.

Erol, K., J. A. Hendler, and D. S. Nau (1994). Umcp: A sound and complete procedure for hierarchical task-network planning. In *AIPS*, Volume 94, pp. 249–254.

Escalante, H. J., M. Montes, and L. E. Sucar (2009). Particle swarm model selection. *Journal of Machine Learning Research 10*(Feb), 405–440.

Falkner, S., A. Klein, and F. Hutter (2018, 10–15 Jul). BOHB: Robust and efficient hyperparameter optimization at scale. In J. Dy and A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning*, Volume 80 of *Proceedings of Machine Learning Research*, Stockholmsmässan, Stockholm Sweden, pp. 1437–1446. PMLR.

Fernández-Delgado, M., E. Cernadas, S. Barro, and D. Amorim (2014). Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research 15*, 3133–3181.

Feurer, M., K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter (2018). Practical automated machine learning for the automl challenge 2018. In *International Workshop on Automatic Machine Learning at ICML*.

Feurer, M., A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter (2015). Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pp. 2962–2970.

Fisher, W. D. (1958). On grouping for maximum homogeneity. *Journal of the American statistical Association 53*(284), 789–798.

Freund, Y. and R. E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences 55*(1), 119–139.

Friedman, J., T. Hastie, and R. Tibshirani (2001). *The elements of statistical learning*, Volume 1. Springer series in statistics New York, NY, USA:.

Friedman, J., T. Hastie, R. Tibshirani, et al. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics 28*(2), 337–407.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 1189–1232.

Fusi, N., R. Sheth, and M. Elibol (2018). Probabilistic matrix factorization for automated machine learning. In *Advances in Neural Information Processing Systems*, pp. 3352–3361.

Geurts, P., D. Ernst, and L. Wehenkel (2006). Extremely randomized trees. *Machine learning 63*(1), 3–42.

Guo, C. and F. Berkhahn (2016). Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*.

Guyon, I., I. Chaabane, H. J. Escalante, S. Escalera, D. Jajetic, J. R. Lloyd, N. Macià, B. Ray, L. Romaszko, M. Sebag, A. Statnikov, S. Treguer, and E. Viegas (2016, 24 Jun). A brief review of the chalearn automl challenge: Any-time any-dataset learning without human intervention. In F. Hutter, L. Kotthoff, and J. Vanschoren (Eds.), *Proceedings of the Workshop on Automatic Machine Learning*, Volume 64 of *Proceedings of Machine Learning Research*, New York, New York, USA, pp. 21–30. PMLR.

Guyon, I. and A. Elisseeff (2006). An introduction to feature extraction. In *Feature extraction*, pp. 1–25. Springer.

H2O.ai (2019a, 1). *AutoML: Automatic Machine Learning*.

H2O.ai (2019b, 1). *H2O*. 3.10.0.8.

Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter 11*(1), 10–18.

Hofner, B., L. Boccuto, and M. Goeker (2015). Controlling false discoveries in high-dimensional situations: Boosting with stability selection. *BMC Bioinformatics 16*(144).

Hofner, B., T. Hothorn, T. Kneib, and M. Schmid (2011). A framework for unbiased model selection based on boosting. *Journal of Computational and Graphical Statistics 20*(4), 956–971.

Hofner, B., A. Mayr, N. Fenske, and M. Schmid (2018). *gamboostLSS: Boosting Methods for GAMLSS Models*. R package version 2.0-1.

Holmes, G., A. Donkin, and I. H. Witten (1994). Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pp. 357–361. IEEE.

Horn, D. and B. Bischl (2016). Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, pp. 1–8. Ieee.

Horn, D., T. Wagner, D. Biermann, C. Weihs, and B. Bischl (2015). Model-based multi-objective optimization: taxonomy, multi-point proposal, toolbox and benchmark. In *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 64–78. Springer.

Hothorn, T., P. Buehlmann, T. Kneib, M. Schmid, and B. Hofner (2010). Model-based boosting 2.0. *Journal of Machine Learning Research 11*, 2109–2113.

Hothorn, T., P. Buehlmann, T. Kneib, M. Schmid, and B. Hofner (2018). *mboost: Model-Based Boosting*. R package version 2.9-1.

Huang, D., T. T. Allen, W. I. Notz, and N. Zeng (2006). Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of global optimization 34*(3), 441–466.

Hutter, F., H. H. Hoos, and K. Leyton-Brown (2011). Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pp. 507–523. Springer.

Jamieson, K. and A. Talwalkar (2016). Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pp. 240–248.

Jeong, S. and S. Obayashi (2005). Efficient global optimization (ego) for multi-objective problem and data mining. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, Volume 3, pp. 2138–2145. IEEE.

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

Jordon, J., J. Yoon, and M. van der Schaar (2019). KnockoffGAN: Generating knockoffs for feature selection using generative adversarial networks. In *International Conference on Learning Representations*.

Kanter, J. M. and K. Veeramachaneni (2015). Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19-21, 2015*, pp. 1–10. IEEE.

Ke, G., Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu (2017). Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pp. 3146–3154.

Keane, A. J. (2006). Statistical improvement criteria for use in multiobjective design optimization. *AIAA journal 44*(4), 879–891.

Kearns, M. and L. Valiant (1994). Cryptographic limitations on learning boolean formulae and finite automata. *Journal of the ACM (JACM) 41*(1), 67–95.

Knowles, J. (2006). Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation 10*(1), 50–66.

Koch, P., B. Bischl, O. Flasch, T. Bartz-Beielstein, C. Weihs, and W. Konen (2012). Tuning and evolution of support vector kernels. *Evolutionary Intelligence 5*(3), 153–170.

Komer, B., J. Bergstra, and C. Eliasmith (2014). Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. In *ICML workshop on AutoML*, pp. 2825–2830.

Kotthoff, L., C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research 18*(1), 826–830.

Kühn, D., P. Probst, J. Thomas, and B. Bischl (2018). Automatic exploration of machine learning experiments on openml. *arXiv preprint arXiv:1806.10961*.

Kursa, M. B., W. R. Rudnicki, et al. (2010). Feature selection with the boruta package. *J Stat Softw 36*(11), 1–13.

Li, L., K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research 18*(1), 6765–6816.

Li, L., K. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, and A. Talwalkar (2018). Massively parallel hyperparameter tuning.

Madrid, J., H. J. Escalante, E. Morales, W.-W. Tu, Y. Yu, L. Sun-Hosoya, I. Guyon, and M. Sebag (2018). Towards automl in the presence of drift: first results. In *International Workshop on Automatic Machine Learning at ICML*.

Mayr, A., N. Fenske, B. Hofner, T. Kneib, and M. Schmid (2012). Generalized additive models for location, scale and shape for high dimensional data—a flexible approach based on boosting. *Journal of the Royal Statistical Society: Series C (Applied Statistics) 61*(3), 403–427.

Mayr, A., B. Hofner, and M. Schmid (2012). The importance of knowing when to stop. *Methods of Information in Medicine 51*(02), 178–186.

Meinshausen, N. and P. Bühlmann (2010). Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 72*(4), 417–473.

Micci-Barreca, D. (2001). A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems. *ACM SIGKDD Explorations Newsletter 3*(1), 27–32.

Mohr, F., M. Wever, and E. Hüllermeier (2018). Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning 107*(8-10), 1495–1515.

Moody, J. (1989). Fast learning in multi-resolution hierarchies. In *Advances in neural information processing systems*, pp. 29–39.

Nargesian, F., H. Samulowitz, U. Khurana, E. B. Khalil, and D. Turaga (2017). Learning feature engineering for classification. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI*, Volume 17, pp. 2529–2535.

Niculescu-Mizil, A. and R. Caruana (2005). Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pp. 625–632. ACM.

Olson, R. S., N. Bartley, R. J. Urbanowicz, and J. H. Moore (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, New York, NY, USA, pp. 485–492. ACM.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research 12*(Oct), 2825–2830.

Picheny, V., D. Ginsbourger, Y. Richet, and G. Caplin (2013). Quantile-based optimization of noisy computer experiments with tunable precision. *Technometrics 55*(1), 2–13.

Probst, P., B. Bischl, and A.-L. Boulesteix (2018). Tunability: Importance of hyperparameters of machine learning algorithms. *arXiv preprint arXiv:1802.09596*.

Probst, P. and A.-L. Boulesteix (2018). To tune or not to tune the number of trees in random forest. *Journal of Machine Learning Research 18*(181), 1–18.

Probst, P., M. Wright, and A.-L. Boulesteix (2018). Hyperparameters and tuning strategies for random forest. *arXiv preprint arXiv:1804.03515*.

Prokhorenkova, L., G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin (2018). Catboost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, pp. 6639–6649.

Rashmi, K. and R. Gilad-Bachrach (2015). Dart: Dropouts meet multiple additive regression trees. In *International Conference on Artificial Intelligence and Statistics*, pp. 489–497.

Rigby, R. A. and D. M. Stasinopoulos (2005). Generalized additive models for location, scale and shape. *Journal of the Royal Statistical Society: Series C (Applied Statistics) 54*(3), 507–554.

Romero, R., J. Espinoza, F. Gotsch, J. Kusanovic, L. Friel, O. Erez, S. Mazaki-Tovi, N. Than, S. Hassan, and G. Tromp (2006). The use of high-dimensional biology (genomics, transcriptomics, proteomics, and metabolomics) to understand the preterm parturition syndrome. *BJOG: An International Journal of Obstetrics & Gynaecology 113*(s3), 118–135.

Saberian, M. J. and N. Vasconcelos (2011). Multiclass boosting: Theory and algorithms. In *Advances in Neural Information Processing Systems*, pp. 2124–2132.

Schapire, R. E. (1990). The strength of weak learnability. *Machine learning 5*(2), 197–227.

Schmid, M., S. Potapov, A. Pfahlberg, and T. Hothorn (2010). Estimation and regularization techniques for regression models with multidimensional prediction functions. *Statistics and Computing 20*(2), 139–150.

Shah, R. D. and R. J. Samworth (2013). Variable selection with error control: another look at stability selection. *Journal of the Royal Statistical Society Series B 75*(1), 55–80.

Snoek, J., H. Larochelle, and R. P. Adams (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*(1), 1929–1958.

Stasinopoulos, M. and R. Rigby (2018). *gamlss.dist: Distributions for Generalized Additive Models for Location Scale and Shape*. R package version 5.1-1.

Strobl, C., A.-L. Boulesteix, A. Zeileis, and T. Hothorn (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC bioinformatics 8*(1), 25.

Sun-Hosoya, L., I. Guyon, and M. Sebag. Activmetal: Algorithm recommendation. In *Workshop on Interactive Adaptive Learning*, pp. 48.

Thornton, C., F. Hutter, H. H. Hoos, and K. Leyton-Brown (2013). Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 847–855. ACM.

Tyree, S., K. Q. Weinberger, K. Agrawal, and J. Paykin (2011). Parallel boosted regression trees for web search ranking. In *Proceedings of the 20th international conference on World wide web*, pp. 387–396. ACM.

Vanschoren, J., J. N. Van Rijn, B. Bischl, and L. Torgo (2014). Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter 15*(2), 49–60.

Wilson, J., A. K. Meher, B. V. Bindu, M. Sharma, V. Pareek, S. Chaudhury, and B. Lall (2018). Autogbt: Automatically optimized gradient boosting trees for classifying large volume high cardinality data streams under concept-drift.

Wistuba, M., N. Schilling, and L. Schmidt-Thieme (2017). Automatic frankensteining: Creating complex ensembles autonomously. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pp. 741–749. SIAM.

Yang, C., Y. Akimoto, D. W. Kim, and M. Udell (2018). Oboe: Collaborative filtering for automl initialization. *arXiv preprint arXiv:1808.03233*.

Zhang, Q. and W. Wang (2007). A fast algorithm for approximate quantiles in high speed data streams. In *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on*, pp. 29–29. IEEE.

Zoph, B. and Q. V. Le (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Zoph, B., V. Vasudevan, J. Shlens, and Q. V. Le (2017). Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012 2*(6).

Appendices

A. Comparison of Gradient Boosting and Random Forest

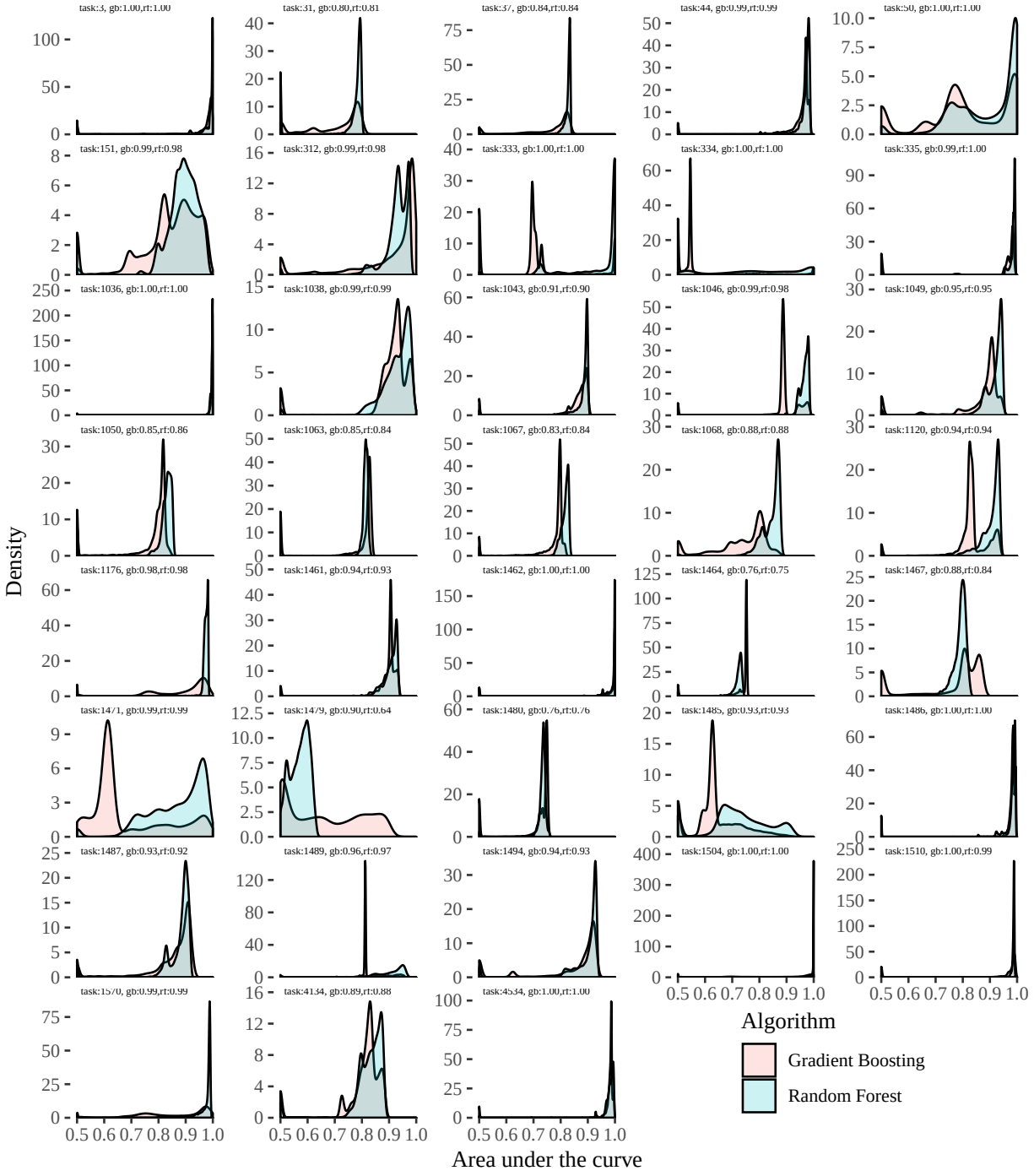

Figure A.1.: Distribution of gradient boosting and random forest performance over 38 datasets. *task* denotes the id of the OpenML task, the numbers after *gb* and *rf* denote the best performance for gradient boosting and the random forest respectively. Overall 50000 evaluations of each algorithm with random hyperparameters are shown. A detailed description of the distribution of evaluations, selection of datasets and hyperparameter ranges can be found in Kühn et al. (2018).

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12. Juli 2011, §8 Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig, ohne unerlaubte Beihilfe angefertigt ist.

München, den 21.02.2019 Janek Thomas