Multi-Objective Optimization of Performance and Interpretability of Tabular Supervised Machine Learning Models

Lennart Schneider
LMU Munich & Munich Center for Machine Learning (MCML)
Munich, Germany
lennart.schneider@stat.uni-muenchen.de

Bernd Bischl
LMU Munich & Munich Center for Machine Learning (MCML)
Munich, Germany
bernd.bischl@stat.uni-muenchen.de

Janek Thomas
LMU Munich & Munich Center for Machine Learning (MCML)
Munich, Germany
janek.thomas@stat.uni-muenchen.de

ABSTRACT

We present a model-agnostic framework for jointly optimizing the predictive performance and interpretability of supervised machine learning models for tabular data. Interpretability is quantified via three measures: feature sparsity, interaction sparsity of features, and sparsity of non-monotone feature effects. By treating hyperparameter optimization of a machine learning algorithm as a multi-objective optimization problem, our framework allows for generating diverse models that trade off high performance and ease of interpretability in a single optimization run. Efficient optimization is achieved via augmentation of the search space of the learning algorithm by incorporating feature selection, interaction and monotonicity constraints into the hyperparameter search space. We demonstrate that the optimization problem effectively translates to finding the Pareto optimal set of groups of selected features that are allowed to interact in a model, along with finding their optimal monotonicity constraints and optimal hyperparameters of the learning algorithm itself. We then introduce a novel evolutionary algorithm that can operate efficiently on this augmented search space. In benchmark experiments, we show that our framework is capable of finding diverse models that are highly competitive or outperform state-of-the-art XGBoost or Explainable Boosting Machine models, both with respect to performance and interpretability.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning**; **Feature selection**.

KEYWORDS

supervised learning, performance, interpretability, tabular data, multi-objective, evolutionary computation, group structure

1 INTRODUCTION

Tabular data are highly relevant for numerous application areas such as finance, bio-informatics, and medical diagnosis. State-of-the-art learning algorithms for tabular data include tree-based methods, e.g., gradient boosted trees (with larger depth) [20] such as XGBoost [8] and LightGBM [33], or random forests [6], which often still outperform deep neural networks [25], although the performance gap has recently shrunk considerably [23, 25, 31, 50]. To achieve peak predictive performance, AutoML tools such as AutoGluon-Tabular [15] or AutoSklearn [19] often make further use of ensembling and stacking multiple models. Moreover, careful hyperparameter optimization of learning algorithms is typically required to yield well performing models [47, 52].

While good predictive performance is generally of central importance, many applications desire or even require models to fulfill additional criteria, such as *interpretability* or *sparseness*. For example a model used for medical diagnosis that achieves high accuracy but lacks interpretability, such as black box models like gradient boosted trees or deep neural networks, may encounter difficulties in gaining trust and adoption. In contrast, a model that can provide insights into its reasoning, even if it has slightly lower performance, is more likely to be trusted and used in real-world scenarios. In the field of Interpretable Machine Learning [41], two different approaches for achieving *interpretability* of models have broadly emerged: **(i)** to only consider learning algorithms that induce "interpretable" models due to their simple intrinsic nature (e.g., logistic regression, decision trees, rule-based systems or generalized additive models) or **(ii)** to use post-hoc methods – which can either be model-agnostic, such as partial dependence plots (PDP) [20] or accumulated local effects (ALE) [1], or model-specific – to gain insight into the inner workings of a model.

When working with tabular data in real-world situations, finding the "right" model can be cumbersome and involves time-consuming manual trial and error. Often, various learning algorithms are tried to produce different models, which are then inspected to select a final model based on concrete user preferences at hand. While this process may be feasible if the goal is to "simply" find a good-performing model, it becomes inefficient if additional criteria such as feature *sparseness*, few *interactions* of features, or *monotonicity* of feature effects are also to be considered. In particular, monotonicity can be highly relevant in practice, as frequently only a model consistent with domain knowledge is acceptable to domain experts. For example, in credit loan approval, models are often required to be monotone with respect to the decision variables involved [53]. Our framework allows automatic generation of a set of models that balance performance and *interpretability*. Formally, this requires two things: **(i)** a way to measure the interpretability of models on a global scale, and **(ii)** an efficient approach for solving the arising *multi-objective* optimization problem.

Our Contributions. We introduce a general, model-agnostic framework for jointly optimizing the predictive performance and interpretability of supervised machine learning models for tabular data. To achieve this, we propose a quantification of the *interpretability* of models on a global scale based on three measures: feature sparsity, interaction sparsity of features, and sparsity of non-monotone feature effects. We then formulate a multi-objective optimization problem of performance and interpretability over the

hyperparameter search space of a learning algorithm, which is augmented by incorporating feature selection as well as interaction and monotonicity constraints into the hyperparameter search space. As a solution to the optimization problem, we present a novel hyperparameter optimization algorithm that can operate efficiently on this augmented search space, making use of the principles of evolutionary computation by treating feature selection as well as the specification of interaction and monotonicity constraints of features as a grouping problem.

2 RELATED WORK

When choosing a learning algorithm that induces interpretable models – e.g., logistic regression models, Elastic-Nets [58], or generalized additive models (GAMs) [28] – one typically loses predictive performance compared to black box models obtained via, e.g., tree based ensembles [10]. However, the downside of these black box models is that their interpretability is hindered by potentially plenty of interaction effects of features and non-linear or non-monotone feature effects. The Explainable Boosting Machine (EBM) [39, 40] positions itself between comparably poor-performing but intelligible models and well-performing but unintelligible models. EBM is a tree-based, cyclic gradient boosting GAM using automatic interaction detection based on FAST [40] to include a given number of second-order interactions in the model. EBM often yields good predictive performance [45] while being more intelligible than black box models. Nevertheless, EBM has some drawbacks: **(i)** EBM is comparably slow to train, as it relies on a large number of boosting steps with a small learning rate to cycle through all features[1], **(ii)** EBM naturally cannot induce a sparse model, as all features are included in a round robin fashion, and the contribution of each feature to a final prediction is therefore non-zero, **(iii)** as a result of the large number of boosting steps, EBM often fits highly non-linear and non-monotone shape functions (resulting in rather complex relationships of features and target), and, relatedly, **(iv)** EBM cannot handle monotonicity constraints during training – i.e., if it is known (or even required) that a feature should have a monotone increasing effect on the target variable, EBM can neither make use of this information nor guarantee such an effect.

A popular approach for constructing sparser models is given by feature selection, which is also related to the complexity and intelligibility of a model [2, 5, 26]. While feature selection can also be performed in the context of unsupervised learning [27], we focus on the supervised learning context. Here, the goal of feature selection is to select only a subset of relevant features while still constructing a model with good predictive performance. There are two model-agnostic approaches to feature selection [26]: feature filters and feature wrappers. Feature filters use proxy measures that are cheap to compute to rank features by their potential explanatory power independent of the concrete learning algorithm being used. Popular examples include measures based on information theory, correlation, distance, or consistency [11]. In contrast to feature filters, feature wrappers directly optimize predictive performance over the space of feature subsets [35]. As every feature subset evaluation requires one or multiple model fits, making exhaustive search infeasible, a discrete black box optimization search strategy (such as a greedy search or an evolutionary algorithm [55]) is necessary. On the one hand, feature selection is often considered a single-objective optimization problem, and the feature selection step is only used to optimize performance [35]. On the other hand, feature selection can also be framed as a multi-objective optimization problem, maximizing predictive performance and feature sparsity simultaneously [2, 54]. Finally, recent work also explored the idea of identifying sets of features without predefined grouping [29].

Looking at measures for interpretability of models on a global scale, Molnar and colleagues [42] were among the first to explicitly propose model-agnostic measures of model complexity. They quantify model complexity by decomposing the prediction function of any model into a sum of components with increasing dimensionality, based on which they derive three measures: the number of features used by a model, the interaction strength of features, and the main effect complexity of features.

3 THEORETICAL BACKGROUND

Consider the supervised learning problem of inferring a model from labeled data $\mathcal{D}$ with n observations where each observation $(\mathbf{x}^{(i)}, y^{(i)})$ consists of a p-dimensional feature vector $\mathbf{x}^{(i)}$. We assume that $\mathcal{D}$ has been sampled i.i.d. from an underlying, unknown distribution, $\mathcal{D} \sim (\mathbb{P}_{xy})^n$. A learning algorithm or *inducer* $\mathcal{I}$ configured by hyperparameters $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ maps a data set $\mathcal{D}$ to a model $\hat{f}$, i.e., $\mathcal{I} : \mathbb{D} \times \boldsymbol{\Lambda} \to \mathcal{H}, (\mathcal{D}, \boldsymbol{\lambda}) \mapsto \hat{f}_{\mathcal{D},\boldsymbol{\lambda}}$, where $\mathbb{D} := \bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n$ is the set of all data sets, $\boldsymbol{\Lambda}$ is the search space of hyperparameters, and $\mathcal{H}$ is the hypothesis space of models. In general, one is interested in constructing a model $\hat{f}_{\mathcal{D},\boldsymbol{\lambda}} = \mathcal{I}(\mathcal{D}, \boldsymbol{\lambda})$ that minimizes the *generalization error*[2], $\text{GE}(\hat{f}_{\mathcal{D},\boldsymbol{\lambda}}) = \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}_{xy}}\left[L(\hat{f}_{\mathcal{D},\boldsymbol{\lambda}}(\mathbf{x}), y)\right]$, where L is a loss function measuring discrepancy between the prediction and true label. However, the generalization error can only be estimated using in-sample data, $\widehat{\text{GE}}(\mathcal{I}_{\boldsymbol{\lambda}}, \mathcal{D})$, through a resampling technique such as cross-validation. For more details, see, e.g., [3, 18].

3.1 Multi-Objective Hyperparameter Optimization

Let $c_1 : \boldsymbol{\Lambda} \to \mathbb{R}, \ldots, c_m : \boldsymbol{\Lambda} \to \mathbb{R}, m \in \mathbb{N}$ denote m evaluation criteria of machine learning models. Note that evaluation criteria usually also depend on the data set and resampling technique at hand (which we omit here for clarity). Define $c : \boldsymbol{\Lambda} \to \mathbb{R}^m$ to assign an m-dimensional cost vector to a hyperparameter configuration $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$. The general multi-objective hyperparameter optimization problem is then defined as $\min_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} c(\boldsymbol{\lambda}) = \min_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} (c_1(\boldsymbol{\lambda}), c_2(\boldsymbol{\lambda}), \ldots, c_m(\boldsymbol{\lambda}))$. Generally, there is no single hyperparameter configuration that minimizes all criteria, as these criteria typically compete with one another. Therefore, focus is given to the concept of Pareto optimality and the set of Pareto optimal configurations: A hyperparameter configuration $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ *(Pareto-)dominates* another configuration $\boldsymbol{\lambda}' \in \boldsymbol{\Lambda}$, written as $\boldsymbol{\lambda} \prec \boldsymbol{\lambda}'$, if and only if

$$\forall i \in \{1, \ldots m\} : c_i(\boldsymbol{\lambda}) \leq c_i(\boldsymbol{\lambda}') \wedge$$
$$\exists j \in \{1, \ldots m\} : c_j(\boldsymbol{\lambda}) < c_j(\boldsymbol{\lambda}').$$

[1] Which we also observed in our benchmark experiments.

[2] With a slight abuse of notation, we will write $\mathcal{I}_{\boldsymbol{\lambda}}$ to denote that a certain hyperparameter configuration $\boldsymbol{\lambda}$ is fixed, i.e., $\mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D}) = \mathcal{I}(\mathcal{D}, \boldsymbol{\lambda})$ with $\boldsymbol{\lambda}$ fixed.

The set of Pareto optimal solutions is therefore defined as $\mathcal{P} := \left\{\boldsymbol{\lambda} \in \boldsymbol{\Lambda} \mid \nexists\, \boldsymbol{\lambda}' \in \boldsymbol{\Lambda} \text{ s.t. } \boldsymbol{\lambda}' \prec \boldsymbol{\lambda}\right\}$. The image of $\mathcal{P}$ under c, $c(\mathcal{P})$, is called the Pareto front. The goal of multi-objective optimization is to find a set of configurations $\hat{\mathcal{P}}$ so that $c(\hat{\mathcal{P}})$ approximates the true Pareto front well.

A popular quality indicator of multi-objective optimization is given by the dominated Hypervolume [57]. The Hypervolume of an approximation of the Pareto front $c(\hat{\mathcal{P}})$ is defined as the combined volume of the dominated hypercubes of all solution points with respect to a reference point $\boldsymbol{r} \in \mathbb{R}^m$. For more details on multi-objective hyperparameter optimization in general as well as an overview of recent applications, we refer to [32, 43].

3.2 Quantifying Interpretability

We propose a quantification of interpretability that is conceptually similar to [42], but our measures and their operationalization differ. As measures for the interpretability of a model on a global scale, we propose to use feature sparsity, interaction sparsity of features, and sparsity of non-monotone features. All our measures are based on the prediction function $\hat{f} : \mathcal{X} \to \mathbb{R}^g$ of a model[3].

To define whether feature j is used by the model, we can determine whether the prediction function changes if the value of x_j changes, i.e., $\hat{f}(x_1, \ldots, x'_j, \ldots x_p) \neq \hat{f}(x_1, \ldots, x_j, \ldots x_p)$ whenever $x'_j \neq x_j$. The (relative) number of features used by a model, NF, can then be defined as

$$NF(\hat{f}) := |\{j \in \{1, \ldots, p\} : \exists x_j, x'_j \in \mathcal{X}_j, x'_j \neq x_j \text{ s.t. } \hat{f}(x_1, \ldots, x'_j, \ldots x_p) \neq \hat{f}(x_1, \ldots, x_j, \ldots x_p)\}|/p. \tag{1}$$

Similarly, we want to define whether two features j and k interact. A prediction function $\hat{f}$ of a model exhibits an interaction between two features j and k if the difference in the value of $\hat{f}(\mathbf{x})$ as a result of changing the value of x_j depends on the concrete value of x_k [21]. Consequently, given no interaction of features j and k, $\hat{f}$ can be decomposed into $\hat{f}(\mathbf{x}) = f_{-j}(\mathbf{x}_{-j}) + f_{-k}(\mathbf{x}_{-k})$ where $\mathbf{x}_{-j}$ and $\mathbf{x}_{-k}$ are feature vectors excluding x_j and respectively x_k. The (relative) number of interactions in a model, NI, can then be defined as

$$NI(\hat{f}) := |\{\{j, k\}, j, k \in \{1, \ldots, p\}, k > j : \nexists f_{-j}, f_{-k} \text{ s.t. } \hat{f}(\mathbf{x}) = f_{-j}(\mathbf{x}_{-j}) + f_{-k}(\mathbf{x}_{-k})\}|/((p(p-1))/2). \tag{2}$$

If the hypothesis space of an inducer is restricted to only contain models including main effects and second-order interaction effects of features, NI is a direct measure of the violation of interaction sparsity of a model. However, if the hypothesis space contains models that include higher order interaction effects, NI falls short in penalizing such higher order interactions. To penalize the inclusion of many pairwise interactions and higher order interactions, we assume transitivity with respect to the interaction of features, i.e., if feature j and k and k and l interact, we also count an interaction of feature j and l.

Finally, we define feature j to have a monotone increasing effect if it holds that whenever $x_j \leq x'_j$, one has that $\hat{f}(x_1, \ldots, x_j, \ldots x_p) \leq \hat{f}(x_1, \ldots, x'_j, \ldots x_p)$. Analogously, we define feature j to have a monotone decreasing effect. The (relative) number of non-monotone features in a model, NNM, is then given by

$$\begin{aligned} NNM(\hat{f}) := |\{j \in \{1, \ldots, p\} : (\exists x_j, x'_j \in \mathcal{X}_j, x_j \leq x'_j \text{ s.t. } \\ \hat{f}(x_1, \ldots, x_j, \ldots x_p) > \hat{f}(x_1, \ldots, x'_j, \ldots x_p)) \wedge \\ (\exists x_j, x'_j \in \mathcal{X}_j, x_j \leq x'_j \text{ s.t. } \\ \hat{f}(x_1, \ldots, x_j, \ldots x_p) < \hat{f}(x_1, \ldots, x'_j, \ldots x_p))\}|/p. \end{aligned} \tag{3}$$

Based on these formal definitions, NF, NI, and NNM can be *operationalized* in different ways. For example, NF can be estimated via a sampling procedure, as described in [42]. Similarly, NI could in principle be estimated based on the partial dependence function [21] or by calculating H-statistics [21] or Greenwell's interaction index [24] for all pairs of features. Depending on the concrete learning algorithm at hand, NF and NI can often also be determined in a straightforward manner by, e.g., looking at features used in splits in a decision tree. In the following, we will exactly determine NF and NI by directly inspecting the resulting model whenever possible. Finally, looking at monotonicity, estimating NNM is arguably difficult. In principle, one could try to test whether a feature has a monotone effect via verification-based testing [49] or adaptive random testing [9]. However, such procedures are always at risk of error, and as monotonicity is typically a hard[4] requirement of a model [46, 53], we opt to determine NNM based on the configuration of the inducer. This requires the inducer to allow for the specification of monotonicity constraints of features, which is easily achievable for, e.g., tree-based methods or GAMs.

We want to note that a model that has low values with respect to NF, NI and NNM still can be complex and must not necessarily result in being intrinsically interpretable. Nevertheless, we believe that such a model is much more easier to interpret, e.g., based on a post-hoc ALE analysis, compared to a model with high values in NF, NI, or NNM. For instance, if a model uses only a few features that have monotone increasing effects and do not interact with each other, the prediction function of the model can be easily summarized. For example, increasing the value of any individual feature would result in an increase in the predicted outcome, regardless of the values of other features. Such a simple and consistent relationship between features and the predicted outcome makes the model more *interpretable*. This direct connection between model *complexity* and ease of interpretability is also the reason why we deem it appropriate to speak of multi-objective optimization of performance and *interpretability*.

3.3 Multi-Objective Optimization of Performance and Interpretability

We formulate the hyperparameter optimization problem of a learning algorithm as a multi-objective optimization problem with the goal of minimizing the estimated generalization error, NF, NI and NNM. To allow for efficient optimization, we extend the search space of the learning algorithm and include hyperparameters for the selection of features, interaction constraints, and monotonicity constraints of features to be part of the search space. Therefore, we

[3] For regression, g is 1, while in classification the output usually represents the g decision scores or posterior probabilities of the g candidate classes. Without loss of generalization, we will assume $g = 1$ in the following.

[4] In practice, a feature is typically expected to exhibit a monotone effect, or not, without any in-between or probabilistic formulation.

require the learning algorithm to allow for the specification of feature selection as well as interaction and monotonicity constraints of features.

In the following, we denote by $\check{\boldsymbol{\Lambda}}$ the extended search space. A hyperparameter configuration $\check{\boldsymbol{\lambda}} \in \check{\boldsymbol{\Lambda}}$ is given by the tuple $(\boldsymbol{\lambda}, \boldsymbol{s}, \boldsymbol{I_s}, \boldsymbol{m_{I_s}})$. Here, $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ is the usual hyperparameter configuration of a learning algorithm, $\boldsymbol{s}$ is a binary vector of length p, indicating selection of features, $\boldsymbol{I_s}$ is a symmetric matrix of dimension $p \times p$ with $(\boldsymbol{I_s})_{jk} = 1$ indicating that features j and k are allowed to interact in a model and 0 indicating otherwise, and $\boldsymbol{m_{I_s}}$ is an integer vector of length p indicating monotonicity constraints of features (-1 for monotone decreasing, 1 for monotone increasing, and 0 for unconstrained[5]).

In principle, we could proceed to try solving the multi-objective optimization problem as given in Equation 4:

$$\min_{\check{\boldsymbol{\lambda}} \in \check{\boldsymbol{\Lambda}}} \left(\widehat{\mathrm{GE}} \left(\mathcal{I}_{\check{\boldsymbol{\lambda}}}, \mathcal{D} \right), NF \left(\hat{f}_{\mathcal{D}, \check{\boldsymbol{\lambda}}} \right), NI \left(\hat{f}_{\mathcal{D}, \check{\boldsymbol{\lambda}}} \right), NNM \left(\hat{f}_{\mathcal{D}, \check{\boldsymbol{\lambda}}} \right) \right) \tag{4}$$

Although this formulation of the optimization problem is quite natural, it has several drawbacks: First, note that the extended search space has become difficult, including a binary vector, a quadratic matrix, and an integer vector that scale linearly or quadratic in the number of features p. Second, note that $\boldsymbol{I_s}$ depends on $\boldsymbol{s}$, as only features that have been selected can be allowed to interact. Similarly, $\boldsymbol{m_{I_s}}$ depends on both $\boldsymbol{I}$ and $\boldsymbol{s}$. For example, if feature j is required to have a monotone increasing effect but is also allowed to interact with another feature k, then the monotonicity of feature j may not be guaranteed if feature k does not also have a monotone increasing effect. This is because the interaction between feature j and k can potentially alter the overall effect of feature j, and without the monotonicity constraint on feature k, the monotonicity of feature j may be compromised. Therefore, in the general model-agnostic case, it is most straightforward to require both features j and k to have monotone increasing effects to ensure that the monotonicity of feature j is maintained in the presence of their potential interaction effect.

We will now derive a reformulation of the search space of the optimization problem stated in Equation 4 that is much easier to handle. To do so, recall the definition of an endorelation and the properties reflexive, symmetric, and transitive. Note that a reflexive, symmetric, and transitive endorelation – also called an equivalence relation – imposes a group structure on a set, i.e., it partitions the set by means of its equivalence classes.

To arrive at an easier formulation of the search space of the optimization problem in Equation 4, we define interactions of features as an endorelation. Let $C = \{1, \ldots, p\}$ denote the index set of features and $C_s \subseteq C$ the index set of features selected for inclusion in a model and define an endorelation R on C_s, $R \subseteq C_s \times C_s$. We say feature j and feature k are *allowed to interact* if the model in principle allows for the inclusion of an (interaction) effect of the two, and write jRk. It follows that R is naturally reflexive and symmetric – i.e., if feature j is allowed to interact with feature k, then the reverse also holds, as the interaction of features is non-directional. However, note that the interaction of features must in fact not be transitive – i.e., even if feature j and k and k and l interact in a model, it must not follow that feature j and l also interact. Nevertheless, from a modeling perspective, it is reasonable to *allow* for features j and l to also interact, partially also due to the potential presence of a three-way interaction, which (in the most general scenario) can only be included in a model if R is closed under transitivity (and the same argument can be made for higher-order interactions)[6]. It is therefore natural to always consider the transitive closure of R, resulting in an equivalence relation. This implies that the equivalence classes induced by R partition the index set of selected features and naturally call for working with a *group structure*. Regarding monotonicity constraints of features, we want to note that monotonicity constraints must simply be defined as attributes of the equivalence classes (for the same reason illustrated earlier: if features are allowed to interact, they should share the same monotonicity constraint).

We can now introduce the group structure space $\boldsymbol{\mathcal{G}}$. Each group structure $\boldsymbol{G} \in \boldsymbol{\mathcal{G}}$ consists of a g-tuple of sets of feature indices with the first set, i.e., group, representing the features that were not selected ($C \setminus C_s$) and all remaining sets resembling the k equivalence classes under the equivalence relation $R \subseteq C_s \times C_s$ of features being *allowed to interact* with each equivalence class also being equipped with a monotonicity attribute. Any group structure can therefore be encoded as follows: $\boldsymbol{G} = (G_1 = C \setminus C_s, G_2 = (E_1, M_{E_1}), \ldots, G_g = (E_k, M_{E_k}))$. Here, $E_k \subseteq C_s$ is an index set containing the indices of features part of the k-th equivalence class under R, and $M_{E_k} \in \{-1, 0, 1\}$ is the monotonicity attribute of the k-th equivalence class. We can now reformulate Equation 4 and introduce the augmented search space $\tilde{\boldsymbol{\Lambda}} = \boldsymbol{\Lambda} \times \boldsymbol{\mathcal{G}}$ by considering the group structure $\boldsymbol{G} \in \boldsymbol{\mathcal{G}}$ instead of $\boldsymbol{s}$, $\boldsymbol{I_s}$, and $\boldsymbol{m_{I_s}}$. The reformulated search space now consists of the Cartesian product of the search space of the learning algorithm, $\boldsymbol{\Lambda}$, and the group structure space $\boldsymbol{\mathcal{G}}$ and each configuration, $\tilde{\boldsymbol{\lambda}}$ of the search space is given by a tuple $(\boldsymbol{\lambda}, \boldsymbol{G})$, which we argue is much easier to optimize. We visualize the components involved in the optimization problem in Figure 1.

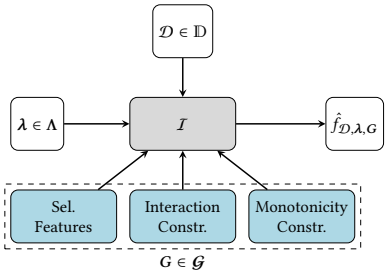

Figure 1: Overview of the components involved in the hyperparameter optimization problem. The inducer is required to allow for the specification of feature selection, as well as interaction and monotonicity constraints of features, which are derived based on the group structure $G \in \mathcal{G}$.

4 METHOD

For optimizing the multi-objective optimization problem, we introduce an optimizer consisting of an evolutionary algorithm (EA) for the original search space of the learning algorithm $\boldsymbol{\Lambda}$ and a

[5]We will later argue that it suffices to only consider $\{0, 1\}$ as monotonicity constraints.

[6]This is also directly related to the principle of marginality; see, e.g., [44].

so-called grouping genetic algorithm (GGA) [16] for the group structure space $\boldsymbol{\mathcal{G}}$. We therefore dub our optimizer *EAGGA*.

4.1 EAGGA

The combination of using an EA and GGA allows us to jointly operate on the augmented search space $\tilde{\boldsymbol{\Lambda}} = \boldsymbol{\Lambda} \times \boldsymbol{\mathcal{G}}$. *EAGGA*'s main routine is heavily inspired by NSGA-II [12]. NSGA-II is an evolutionary multi-objective algorithm making use of the concepts of non-dominated sorting and crowding distance to select individuals for survival close to the Pareto front that also cover a wide spread along the Pareto front. In each generation, NSGA-II iterates through reproduction, crossover, mutation, and survival steps that generate the population of the next generation. In *EAGGA*, we perform parent selection via a binary tournament selection and simply apply suitable crossover and mutation operators to hyperparameters of the original search space ($\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$) and group structures ($\boldsymbol{G} \in \boldsymbol{\mathcal{G}}$) next to each other to produce offspring.

4.1.1 EA Operators. For the original hyperparameters of the learning algorithm ($\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$), we use the Cartesian product of operators that operate in different ways on the different parameter types [37]. We use a global crossover probability of $p = 0.7$ and a global mutation probability of $p = 0.3$. All hyperparameters undergo uniform crossover ($p = 0.5$) for recombination. Numeric and integer hyperparameters undergo Gaussian mutation ($p = 0.2, \sigma = 0.1$; values min-max scaled to $[0, 1]$ prior to mutation and re-transformed afterwards; values rounded to the closest integer in the case of integer hyperparameters), while categorical hyperparameters undergo uniform mutation ($p = 0.2$). The choice of operators and probabilities of crossover and mutation were mostly inspired by [2].

4.1.2 GGA Operators. Group structures ($\boldsymbol{G} \in \boldsymbol{\mathcal{G}}$) undergo mutation and crossover operators inspired by the original work of Falkenauer [16, 17]. We again use a global crossover probability of $p = 0.7$ and a global mutation probability of $p = 0.3$. Recall that a group structure is encoded as $\boldsymbol{G} = (G_1 = C \setminus C_s, G_2 = (E_1, M_{E_1}), \ldots, G_g = (E_k, M_{E_k}))$ where $G_1 = C \setminus C_s$ is an index set of features not selected and each $E_k \subseteq C_s$ is an index set of features part of the k-th equivalence class under the equivalence relation R of features being *allowed to interact*, and $M_{E_k} \in \{-1, 0, 1\}$ is the monotonicity attribute of the k-th equivalence class. The basic idea of a GGA is to apply operators directly on the group structure. For crossover, we select two crossing sites, delimiting the crossing section, in each of the two parents (e.g., $G_1G_2|G_3|G_4$ and $H_1|H_2H_3|H_4H_5$; G used for the first parent and H for the second parent). We then inject the contents (groups together with their monotonicity attributes) of the crossing section of the first parent at the first crossing site of the second parent (e.g., inserting G_3 into the second parent, resulting in $H_1G_3H_2H_3H_4H_5$). Finally, we remove all items (feature indices) from the old groups now occurring twice in the second parent. For example, assume $H_3 = (\{1, 2, 3\}, 0)$ and $G_3 = (\{3\}, 1)$, then after inserting G_3 into the second parent, H_3 is given by $(\{1, 2\}, 0)$. In the case of the first group, i.e., the index set of features not selected, being injected, we simply add these indices to the first group of the parent. To create the second offspring, we swap the roles of the parents. For more details on the GGA crossover, see [17]. For mutation, we simply assign each feature index a new group membership with probability $p = 0.2$ and sample a new monotonicity attribute for each group with probability $p = 0.2$. To allow for more precise handling of the group structure, we incorporate a feedback loop into *EAGGA*: After evaluating an offspring, we can determine the actual features and interactions (closed under transitivity) as included in the model[7] and update the group structure $\boldsymbol{G}$ of each offspring. In Section 5.3 and the supplementary material, we present results of an ablation study investigating the effect of turning off either crossover or mutation of group structures or both, where we observed that in general both of them are needed for good performance.

4.2 Initializing the Group Structures

As hyperparameter optimization is costly, we strive to make *EAGGA* more sample-efficient. To help us to initialize the population's group structures in a more sophisticated manner than a simple random initialization, we introduce three types of so-called *detectors*: feature, interaction, and monotonicity detectors. In Section 5.3 and the supplementary material, we present results of an ablation study investigating the effect of using detectors to initialize the population within *EAGGA*, where we observe that using detectors substantially boosts the anytime performance of *EAGGA*.

4.2.1 Feature Detector. The goal of a *feature detector* is to quantify the importance of features so that the probability of selecting an important feature j (i.e., $j \in C_s$) can be increased. Formally, a feature detector maps a data set $\mathcal{D}$ to a p-dimensional vector of real valued scores with the j-th element corresponding to the score of the j-th feature. In *EAGGA*, we use feature filters. A feature filter quantifies the importance of each feature via a quick-to-compute proxy, e.g., the entropy-based information gain feature filter [36] considers the difference between the sum of the Shannon entropy for the target variable and feature and the joint Shannon entropy for the target variable conditioned on the feature. Based on the filter score for each feature, we can then weight the probability of selecting a feature. To determine the number of selected features S of a member of the initial population, we sample a random integer between 1 and p from a truncated geometric distribution similarly as in [2]. The features that are actually selected are then determined by sampling from all binary vectors $\boldsymbol{s}$ of length p that sum to S with weighted probabilities according to the feature filter scores.

4.2.2 Interaction Detector. The idea of a (pairwise) *interaction detector* is to quantify the importance of interactions of features so that the probability of those features being in the same group (i.e., the same equivalence class under the equivalence relation R *allowed to interact*) can be increased. Formally, an interaction detector maps a data set $\mathcal{D}$ to a symmetric, real valued $p \times p$ matrix with the element at the j-th row and k-th column corresponding to the score of the j-th and k-th feature[8]. Recall that in *EAGGA*, the first group G_1 of a group structure $\boldsymbol{G}$ is always given by the indices of features that are not selected. To initialize the remaining groups, we make use of the FAST algorithm [40]. FAST allows for efficient quantification of

[7] The group structure only imposes an upper constraint, meaning that the resulting model may use all or some of the selected features, and the same applies to interactions.
[8] Note that the diagonal is of no interest and can be set to, e.g., 0.

the importance of all pairwise interactions of features based on the residual sums of squares when extending a main effects model to include an interaction effect. To determine the number of included interactions I of a member of the initial population, we sample a random integer between 1 and $(p(1-p))/2$ from a truncated geometric distribution. The actual groups are then determined by considering the I most important pairwise interactions according to FAST, constructing an equivalence relation R *allowed to interact*, and deriving the equivalence classes under R.

4.2.3 Monotonicity Detector. Using a *monotonicity detector* is helpful due to two reasons: First, recall that the monotonicity attribute of a group can in principle either be -1 (monotone decreasing), 1 (monotone increasing), or 0 (unconstrained). This is somewhat redundant, as a monotone decreasing feature effect (without loss of generalization, we assume purely numeric features) can always be realized by enforcing a monotone increasing effect and swapping the sign of the feature itself. Therefore, by detecting whether a monotone feature effect should be increasing or decreasing we can encode monotonicity constraints more efficiently. Second, by quantifying the mismatch in model fit between enforcing monotonicity and no constraint, the monotonicity detector can bias the probability of the monotonicity attribute being unconstrained. Formally, a monotonicity detector maps a data set $\mathcal{D}$ to a p-dimensional vector of real valued scores with the j-th element corresponding to the score of the j-th feature where the sign of the score indicates the direction of monotonicity and the magnitude of the score reflects the strength of the monotone relationship between the feature and the target variable. In *EAGGA*, we use the following monotonicity detector: For each feature, we fit a decision tree on sub-sampled data and obtain the predictions. We then calculate Spearman's ρ between the feature values and the target predictions. Finally, we repeat this process 10 times and calculate the average Spearman's ρ, which we scale[9] to $[0.2, 0.8]$. For each group of features of a member of the initial population, we take the average over the individual scores and use this average as a probability to sample the monotonicity attribute of the group.

5 BENCHMARK EXPERIMENTS

To our best knowledge, *EAGGA* is the first model-agnostic approach to perform *efficient* multi-objective optimization of performance and interpretability of machine learning models by incorporating feature selection as well as interaction and monotonicity constraints into the hyperparameter search space. In our experiments, we combine *EAGGA* with XGBoost ($EAGGA_{\text{XGBoost}}$) or XGBoost with a maximum depth fixed to 2 ($EAGGA_{\text{XGBoost}_{\text{md2}}}$, resulting in second-order interactions being the most complex higher-order interactions that can be picked up by the model). We configure *EAGGA* to use a population size of $\mu = 100$ and an offspring size of $\nu = 10$, with the comparably large population size being inspired by [2, 54]. One naïve approach to generate a benchmark baseline is to simply use a collection of competitors that all excel at different objectives which *EAGGA* tries to optimize jointly and compare $EAGGA_{\text{XGBoost}}$ to the union of the competitors. Another approach is to compare $EAGGA_{\text{XGBoost}}$ to standard multi-objective optimization of XGBoost (without augmentation of the search space). Code and supplementary material are released via https://github.com/slds-lmu/paper_2023_eagga.

5.1 *EAGGA* vs. A Collection of Competitors

We construct a collection of competitors by considering an EBM, Elastic-Net, (untuned) random forest, and XGBoost. The reasoning behind this is that an EBM should result in good performance using few interaction effects, an Elastic-Net should find a sparse solution with all features having a monotone effect, whereas a random forest and XGBoost should result in good performance but potentially use many features, as well as interactions and non-monotone feature effects. We tune the hyperparameters of the EBM, Elastic-Net, and XGBoost via Bayesian Optimization[10] and optimize for predictive performance. For the search spaces of the learning algorithms, see our supplementary material. All learning algorithms are given a budget of 8 hours of sequential runtime on a single CPU (note that this is a disadvantage for *EAGGA*, as each competitor is given the same computational budget and therefore the union of competitors uses substantially more compute budget than *EAGGA*). As a performance metric, we choose the area under the receiver operating characteristic curve (AUC)[11]. Performance estimation is conducted via nested resampling: As an outer resampling, we use a holdout with a ratio of 2/3, i.e., test performance is evaluated on 1/3 of the data. Hyperparameter optimization is then performed using 5-fold cross-validation on the remaining 2/3 of the data. For $EAGGA_{\text{XGBoost}}$ and $EAGGA_{\text{XGBoost}_{\text{md2}}}$, the Pareto optimal configurations found during optimization are re-evaluated on the test-set. For the EBM, Elastic-Net, random forest, and XGBoost, we re-evaluate the single best-performing configuration (found during optimization) on the test-set. For XGBoost models, NF and NI are determined by actually checking the model and all splits in all trees, whereas NNM is determined based on the monotonicity constraints of features used in the model (only applicable when optimized via *EAGGA*; for the standard XGBoost, we assume NNM to be the same as NF as we consider monotonicity of features to be a hard requirement as explained in Section 3.2). For the EBM, NF is always 1, as EBM cycles through all available features in a round robin fashion, whereas NI is directly given by the value of the hyperparameter `interactions` and we assume NNM to be the same as NF, as EBM does not allow for the specification of monotonicity constraints and cannot guarantee monotone feature effects. For the Elastic-Net, NF is determined by looking at the relative number of non-zero coefficients, whereas NI and NNM are always 0 (no interaction effects are included in the standard Elastic-Net and feature effects are always monotone). Finally, for the random forest, NF and NI are again determined by actually checking the model and all splits in all trees, whereas NNM is again the same as NF (for the same reason as for the standard XGBoost).

All methods are compared on twenty binary classification tasks taken from OpenML CC-18 [4] and the AutoML benchmark [22].

[9] This is done to allow for some non-determinism during sampling.

[10] We employ a Bayesian Optimization variant similarly configured as SMAC [38], i.e., using a random forest as surrogate model and Expected Improvement [30] as acquisition function.

[11] We minimize the negative AUC.

We perform 10 replications of each optimization run on each task with different random seeds to allow for statistical analysis. Criteria for selecting the tasks were fewer than 100000 observations, the number of features being fewer than 1000 as well as numeric features, i.e., we focus on small- to medium-sized tabular data sets. We only consider binary classification tasks, as the EBM until now does not support the inclusion of interaction effects of features in the case of multi-class classification. More details on the data sets can be found in our supplementary material.

As we are comparing a multi-objective optimization framework (*EAGGA*) to a collection of models, we perform the following analysis: For every run on each task, we calculate the dominated Hypervolume of the (test-set) Pareto front of $EAGGA_{\text{XGBoost}}$ and $EAGGA_{\text{XGBoost}_{\text{md2}}}$ with respect to the reference point $\boldsymbol{r} = (0, 1, 1, 1)^\top$ and compare this with the dominated Hypervolume obtained by considering the non-dominated set of the EBM, Elastic-Net, random forest, and XGBoost solutions (evaluated on the test-set). To allow for a fair comparison, we always include a featureless learner that simply predicts the majority class without relying on any features when calculating the dominated Hypervolume[12]. Results are given in Figure 2. Note that the number in parentheses after a task name indicates the number of features of the task. Using *EAGGA* results in substantially larger dominated Hypervolume (Wilcoxon signed-ranks test [13] on the mean dominated Hypervolume over replications: $T = 0, p < 0.001$ for $EAGGA_{\text{XGBoost}}$ vs. competitors and $T = 0, p < 0.001$ for $EAGGA_{\text{XGBoost}_{\text{md2}}}$ vs. competitors).

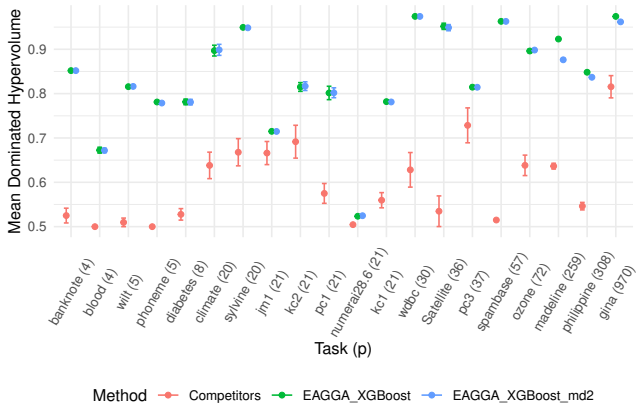

Figure 2: Mean dominated Hypervolume of $EAGGA_{\text{XGBoost}}$, $EAGGA_{\text{XGBoost}_{\text{md2}}}$, and the union of competitors averaged over 10 replications. Bars represent standard errors.

We further determine for each task the fraction of replications where each competitor yields a solution that is Pareto-dominated by the solutions of $EAGGA_{\text{XGBoost}}$ or $EAGGA_{\text{XGBoost}_{\text{md2}}}$. Table 1a shows this fraction averaged over all tasks for $EAGGA_{\text{XGBoost}}$ – i.e., on average, roughly 46% of the EBM solutions are Pareto-dominated by the solutions found by $EAGGA_{\text{XGBoost}}$. Table 1b shows this fraction averaged over all tasks for $EAGGA_{\text{XGBoost}_{\text{md2}}}$. We also compute the counterpart – i.e., what is the fraction of replications where the whole Pareto set of $EAGGA_{\text{XGBoost}}$ or $EAGGA_{\text{XGBoost}_{\text{md2}}}$ is dominated by the Pareto set of the union of the competitors. This was never the case, neither for $EAGGA_{\text{XGBoost}}$ nor $EAGGA_{\text{XGBoost}_{\text{md2}}}$. We want to note that in some runs, evaluating the initial design during optimization of the EBM took longer than the whole compute budget of 8 hours. In these cases, our fallback was to only evaluate the default configuration suggested by the EBM authors.

In our supplementary material, we also provide an illustrative example of the usage of *EAGGA* relying on the *ozone-level-8hr* task and analyze an exemplary Pareto front. Additionally we analyze the best performing models from each method in terms of AUC and interpretability. Results show that the best models found by *EAGGA* perform similarly to XGBoost models optimized for performance, but use less features, interactions, and non-monotone features, indicating improved interpretability.

Table 1: Mean fraction of runs over tasks and replications where competitors yield a solution that is dominated by $EAGGA_{\text{XGBoost}}$ or $EAGGA_{\text{XGBoost}_{\text{md2}}}$.

(a) $EAGGA_{\text{XGBoost}}$

Competitor	Mean	SE
EBM	0.46	0.04
Elastic-Net	0.30	0.03
Random Forest	0.81	0.03
XGBoost	0.40	0.03

SE = standard error.

(b) $EAGGA_{\text{XGBoost}_{\text{md2}}}$

Competitor	Mean	SE
EBM	0.36	0.03
Elastic-Net	0.28	0.03
Random Forest	0.74	0.03
XGBoost	0.31	0.03

SE = standard error.

5.2 *EAGGA* vs. Multi-Objective XGBoost

We also compare $EAGGA_{\text{XGBoost}}$ to multi-objective optimization of XGBoost (without augmentation of the search space), which we will refer to as $\text{XGBoost}_{\text{MO}}$. As an optimizer, we employ ParEGO [34], a scalarization-based multi-objective Bayesian Optimization algorithm that we configure to use a random forest as surrogate model and Expected Improvement as acquisition function. The search space used within ParEGO is exactly the same as the search space used within *EAGGA* – with the exception that we do not augment the search space to include feature selection, interaction, and monotonicity constraints, as standard multi-objective optimizers such as ParEGO cannot naturally operate on such a search space. The question we want to answer is whether it is sufficient to work on the standard search space with a standard multi-objective optimizer to optimize XGBoost for predictive performance and interpretability. Benchmark tasks and the evaluation protocol are exactly the same as in Section 5.1 – i.e., for $EAGGA_{\text{XGBoost}}$, $EAGGA_{\text{XGBoost}_{\text{md2}}}$, and $\text{XGBoost}_{\text{MO}}$, the Pareto optimal configurations found during optimization are re-evaluated on the test-set. For each run on each task, we calculate the dominated Hypervolume of the (test-set) Pareto front of $EAGGA_{\text{XGBoost}}$, $EAGGA_{\text{XGBoost}_{\text{md2}}}$, and $\text{XGBoost}_{\text{MO}}$, which we visualize in Figure 3. Again, using *EAGGA* results in usually at least the same and often substantially larger dominated Hypervolume (Wilcoxon signed-ranks test on the mean dominated Hypervolume over replications: $T = 40, p = 0.0076$ for $EAGGA_{\text{XGBoost}}$ vs. $\text{XGBoost}_{\text{MO}}$ and $T = 50, p = 0.02$ for $EAGGA_{\text{XGBoost}_{\text{md2}}}$ vs.

[12] As the resulting point $(-0.5, 0, 0, 0)^\top$ will have a large contribution to the dominated Hypervolume, but only *EAGGA* might be able to consistently find a hyperparameter configuration resulting in such a model.

$\text{XGBoost}_{\text{MO}}$). Notably, the only tasks where $\text{XGBoost}_{\text{MO}}$ outperforms *EAGGA* are tasks with few features. In our supplementary material, we also analyze the anytime dominated Hypervolume during optimization (i.e., calculated on the inner resampling).

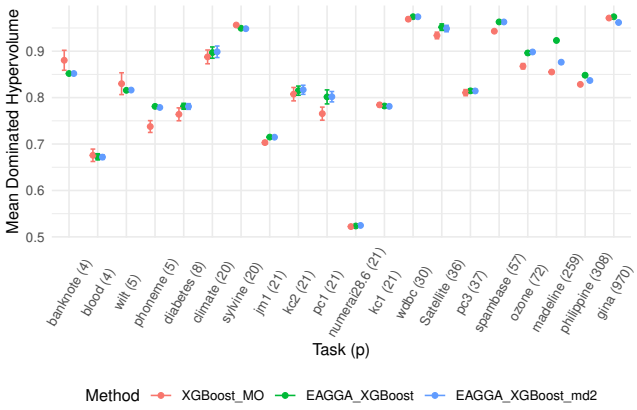

Figure 3: Mean dominated Hypervolume of $\textit{EAGGA}_{\text{XGBoost}}$, $\textit{EAGGA}_{\text{XGBoost}_{\text{md2}}}$, and $\text{XGBoost}_{\text{MO}}$ averaged over 10 replications. Bars represent standard errors.

5.3 An Ablation Study of *EAGGA*

We perform an ablation study of the components of *EAGGA* with the goal to answer the following questions: **(i)** Does *EAGGA* improve over a random search on the same search space? **(ii)** How important are crossover and respectively mutation of group structures? **(iii)** What is the benefit of using detectors to initialize the population?

To do so, we rerun all benchmark experiments with different flavors of *EAGGA* and analyze the mean dominated Hypervolume during optimization, i.e., calculated on the inner resampling. We consider the following modifications or "flavors" of *EAGGA*: **(i)** Simply performing a random search on $\tilde{\mathbf{\Lambda}}$ after using *EAGGA*'s detectors to initialize the population (`Random Search`). **(ii)** Switching off either crossover or mutation of group structures ($\boldsymbol{G} \in \boldsymbol{\mathcal{G}}$) or both (`No_Crossover`, `No_Mutation`, `No_Cross_Mut`). **(iii)** Switching off the detectors of *EAGGA* and initializing the population at random (`No_Detectors`).

We observe that **(i)** performing a random search performs comparably poorly, **(ii)** crossover and mutation of group structures are needed for good performance and **(iii)** using detectors can boost the performance although this is mainly due to using detectors strongly affecting the early performance of *EAGGA*. Conducting a Friedman test [13] on the final mean dominated Hypervolume during optimization indicates significant differences in ranks of optimizers ($\chi^2(6) = 52.99, p < 0.001$). Figure 4 visualizes the corresponding critical difference plot based on the follow up Nemenyi test. For completeness, we also include $\text{XGBoost}_{\text{MO}}$. For detailed results and discussion, please see our supplementary material.

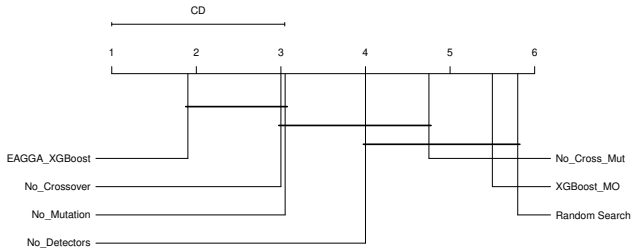

Figure 4: Critical difference plot of the ranks of optimizers based on the final mean dominated Hypervolume during optimization. Lower rank is better.

6 CONCLUSION

We have presented a general model-agnostic framework for jointly optimizing the predictive performance and interpretability of supervised machine learning models for tabular data. *EAGGA* is a multi-objective optimizer making use of the principles of evolutionary computation to jointly optimize the hyperparameters of a learning algorithm as well as the group structure of features. *EAGGA* allows for obtaining a set of diverse models in a single optimization run and can outperform state-of-the-art competitors both with respect to performance and interpretability.

In practice, users may already have prior knowledge about which features to include, which features should interact or even a requirement for a certain feature to have a monotone effect. While we only investigated the scenario of optimizing the group structure of features without prior knowledge, i.e., imposing no restrictions on the group structure, *EAGGA* can be extended to scenarios that involve prior knowledge in a straightforward manner. This can be achieved by, for example, initializing the population in a way to reflect prior knowledge and prohibiting crossover and mutation of group structures to produce offspring that are no longer in congruence to the prior.

EAGGA might be especially useful when using deep neural networks as learning algorithms, as Kadra and colleagues [31] demonstrated that strong regularization of neural networks can be a key component to achieving good performance on tabular data. Using *EAGGA* in combination with neural networks would require the design of a network architecture that allows for the specification of interaction and monotonicity constraints of features. Notable work in this direction has been undertaken by [7, 14, 48, 51, 56].

Finally, it must be noted that *EAGGA* cannot guarantee that the resulting group structure of a model is sensible, and the structure must be verified by domain experts (with respect to the selection of features, as well as their interaction and monotonicity constraints). Nevertheless, we believe that *EAGGA* can be of significant interest for a wide variety of users.

ACKNOWLEDGMENTS

The authors of this work take full responsibilities for its content. Lennart Schneider is supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy through the Center for Analytics - Data - Applications (ADACenter) within the

framework of BAYERN DIGITAL II (20-3410-2-9-8). Lennart Schneider acknowledges funding from the LMU Mentoring Program of the Faculty of Mathematics, Informatics and Statistics.

REFERENCES

[1] D. W. Apley and J. Zhu. 2020. Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 82, 4 (2020), 1059–1086.

[2] M. Binder, J. Moosbauer, J. Thomas, and B. Bischl. 2020. Multi-Objective Hyperparameter Tuning and Feature Selection Using Filter Ensembles. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 471–479.

[3] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, D. Deng, and M. Lindauer. 2021. Hyperparameter Optimization: Foundations, Algorithms, Best Practices, and Open Challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (2021), e1484.

[4] B. Bischl, G. Casalicchio, M. Feurer, P. Gijsbers, F. Hutter, M. Lang, R. Gomes Mantovani, J. N. van Rijn, and J. Vanschoren. 2021. OpenML Benchmarking Suites. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, J. Vanschoren and S. Yeung (Eds.), Vol. 1.

[5] B. Bischl, I. Vatolkin, and M. Preuss. 2010. Selecting Small Audio Feature Sets in Music Classification by Means of Asymmetric Mutation. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I*. 314–323.

[6] L. Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

[7] C.-H. Chang, R. Caruana, and A. Goldenberg. 2022. Node-GAM: Neural Generalized Additive Model for Interpretable Deep Learning. *The Tenth International Conference on Learning Representations, ICLR* (2022).

[8] T. Chen 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*. 785–794.

[9] T. Y. Chen, H. Leung, and I. K. Mak. 2005. Adaptive Random Testing. In *Advances in Computer Science - ASIAN 2004. Higher-Level Decision Making*, M. J. Maher (Ed.). 320–329.

[10] R. Couronné, P. Probst, and A.-L. Boulesteix. 2018. Random Forest versus Logistic Regression: A Large-Scale Benchmark Experiment. *BMC Bioinformatics* 19, 1 (2018), 1–14.

[11] M. Dash and H. Liu. 1997. Feature Selection for Classification. *Intelligent Data Analysis* 1, 3 (1997), 131–156.

[12] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197.

[13] J. Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *The Journal of Machine Learning Research* 7 (2006), 1–30.

[14] A. Dubey, F. Radenovic, and D. Mahajan. 2022. Scalable Interpretability via Polynomials. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35.

[15] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. In *7th ICML Workshop on Automated Machine Learning*.

[16] E. Falkenauer. 1993. The Grouping Genetic Algorithms: Widening the Scope of the GA's. *Belgian Journal of Operations Research, Statistics, and Computer Science* 33, 1–2 (1993), 79–102.

[17] E. Falkenauer. 1996. A Hybrid Grouping Genetic Algorithm for Bin Packing. *Journal of Heuristics* 2, 1 (1996), 5–30.

[18] M. Feurer and F. Hutter. 2019. Hyperparameter Optimization. *Automated Machine Learning: Methods, Systems, Challenges* (2019), 3–33.

[19] M. Feurer, 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*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 28.

[20] J. H. Friedman. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29, 5 (2001), 1189–1232.

[21] J. H. Friedman and B. E. Popescu. 2008. Predictive Learning via Rule Ensembles. *The Annals of Applied Statistics* 2, 3 (2008), 916–954.

[22] P. Gijsbers, M. L. P. Bueno, S. Coors, E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren. 2022. AMLB: An AutoML Benchmark. *arXiv:2207.12560 [cs.LG]* (2022).

[23] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko. 2021. Revisiting Deep Learning Models for Tabular Data. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34.

[24] B. M. Greenwell, B. C. Boehmke, and A. J. McCarthy. 2018. A Simple and Effective Model-Based Variable Importance Measure. *arXiv:1805.04755 [stat.ML]* (2018).

[25] L. Grinsztajn, E. Oyallon, and G. Varoquaux. 2022. Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data?. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[26] I. Guyon and A. Elisseeff. 2003. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research* 3 (2003), 1157–1182.

[27] J. Handl and J. Knowles. 2006. Feature Subset Selection in Unsupervised Learning via Multiobjective Optimization. *International Journal of Computational Intelligence Research* 2, 3 (2006), 217–238.

[28] T. Hastie and R. Tibshirani. 1986. Generalized Additive Models. *Statistical Science* 1, 3 (1986), 297–310.

[29] F. Imrie, A. Norcliffe, P. Liò, and M. van der Schaar. 2022. Composite Feature Selection using Deep Ensembles. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35.

[30] D. R. Jones, M. Schonlau, and W. J. Welch. 1998. Efficient Global Optimization of Expensive Black-Box Functions. *Journal of Global Optimization* 13, 4 (1998), 455–492.

[31] A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka. 2021. Well-tuned Simple Nets Excel on Tabular Datasets. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34.

[32] F. Karl, T. Pielok, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, L. Schneider, J. Thomas, J. Richter, M. Lang, E. C. Garrido-Merchán, J. Branke, and B. Bischl. 2022. Multi-Objective Hyperparameter Optimization - An Overview. *arXiv:2206.07438 [cs.LG]* (2022).

[33] G. Ke, 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*, I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30.

[34] J. Knowles. 2006. ParEGO: A Hybrid Algorithm with On-Line Landscape Approximation for Expensive Multiobjective Optimization Problems. *IEEE Transactions on Evolutionary Computation* 10, 1 (2006), 50–66.

[35] R. Kohavi and G. H. John. 1997. Wrappers for Feature Subset Selection. *Artificial Intelligence* 97, 1–2 (1997), 273–324.

[36] C. Largeron, C. Moulin, and M. Géry. 2011. Entropy Based Feature Selection for Text Categorization. In *Proceedings of the 2011 ACM Symposium on Applied Computing*. 924–928.

[37] R. Li, M. T. M. Emmerich, J. Eggermont, T. Bäck, M. Schütz, J. Dijkstra, and J. H. C. Reiber. 2013. Mixed Integer Evolution Strategies for Parameter Optimization. *Evolutionary Computation* 21, 1 (2013), 29–64.

[38] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research* 23 (2022), 54–1.

[39] Y. Lou, R. Caruana, and J. Gehrke. 2012. Intelligible Models for Classification and Regression. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 150–158.

[40] Y. Lou, R. Caruana, J. Gehrke, and G. Hooker. 2013. Accurate Intelligible Models with Pairwise Interactions. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 623–631.

[41] C. Molnar. 2022. *Interpretable Machine Learning* (2 ed.). https://christophm.github.io/interpretable-ml-book

[42] C. Molnar, G. Casalicchio, and B. Bischl. 2020. Quantifying Model Complexity via Functional Decomposition for Better post-hoc Interpretability. In *Machine Learning and Knowledge Discovery in Databases*, P. Cellier and K. Driessens (Eds.). Springer International Publishing, 193–204.

[43] A. Morales-Hernández, I. van Nieuwenhuyse, and S. Rojas Gonzalez. 2022. A Survey on Multi-Objective Hyperparameter Optimization Algorithms for Machine Learning. *Artificial Intelligence Review* (2022), 1–51.

[44] J. A. Nelder. 1977. A Reformulation of Linear Models. *Journal of the Royal Statistical Society. Series A (General)* 140, 1 (1977), 48–77.

[45] H. Nori, S. Jenkins, P. Koch, and R. Caruana. 2019. InterpretML: A Unified Framework for Machine Learning Interpretability. *arXiv:1909.09223 [cs.LG]* (2019).

[46] R. Potharst and A. J. Feelders. 2002. Classification Trees for Problems with Monotonicity Constraints. *ACM SIGKDD Explorations Newsletter* 4, 1 (2002), 1–10.

[47] P. Probst, A.-L. Boulesteix, and B. Bischl. 2019. Tunability: Importance of Hyperparameters of Machine Learning Algorithms. *Journal of Machine Learning Research* 20, 53 (2019), 1–32.

[48] F. Radenovic, A. Dubey, and D. Mahajan. 2022. Neural Basis Models for Interpretability. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35.

[49] A. Sharma and H. Wehrheim. 2020. Testing Monotonicity of Machine Learning Models. *arXiv:2002.12278 [cs.LG]* (2020).

[50] R. Shwartz-Ziv and A. Armon. 2022. Tabular Data: Deep Learning is Not All You Need. *Information Fusion* 81 (2022), 84–90.

[51] M. Tsang, H. Liu, S. Purushotham, P. Murali, and Y. Liu. 2018. Neural Interaction Transparency (NIT): Disentangling Learned Interactions for Improved

Interpretability. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31.

[52] J. N. van Rijn and F. Hutter. 2018. Hyperparameter Importance Across Datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2367–2376.

[53] M. Velikova and H. A. M. Daniels. 2004. Decision Trees for Monotone Price Models. *Computational Management Science* 1 (2004), 231–244.

[54] B. Xue, W. Fu, and M. Zhang. 2014. Multi-Objective Feature Selection in Classification: A Differential Evolution Approach. In *Simulated Evolution and Learning: 10th International Conference*. 516–528.

[55] B. Xue, M. Zhang, W. N. Browne, and X. Yao. 2016. A Survey on Evolutionary Computation Approaches to Feature Selection. *IEEE Transactions on Evolutionary Computation* 20, 4 (2016), 606—-626.

[56] Z. Yang, A. Zhang, and A. Sudjianto. 2021. GAMI-Net: An Explainable Neural Network Based on Generalized Additive Models with Structured Interactions. *Pattern Recognition* 120 (2021), 108192.

[57] E. Zitzler and L. Thiele. 1998. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*. 292–304.

[58] H. Zou and T. Hastie. 2005. Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67, 2 (2005), 301–320.

A ILLUSTRATIVE EXAMPLE

We illustrate the potential of our approach using a concrete example, relying on the *ozone-level-8hr* task. The goal is to predict whether a day is a high ozone day or not using 72 features such as temperature measured at different time throughout the day. We again use *EAGGA* with XGBoost and compare it to an EBM, Elastic-Net, random forest, and XGBoost. The evaluation protocol is the same as used in the benchmark experiments. We summarize the results in Table 2. *EAGGA* is able to find a good spread of models that trade off performance and interpretability to varying degree. For example, the best-performing XGBoost model found by *EAGGA* is close to the performance of an XGBoost model solely optimized for performance but uses substantially fewer features, interactions, and non-monotone features.

Table 2: Pareto front obtained using $EAGGA_{XGBoost}$ on the *ozone-level-8hr* task compared to the solutions found using an EBM, Elastic-Net, random forest, or XGBoost.

(a) $EAGGA_{XGBoost}$

AUC	NF	NI	NNM
0.802	0.014	0.000	0.000
0.818	0.083	0.002	0.000
0.829	0.139	0.005	0.000
0.831	0.042	0.000	0.028
0.841	0.042	0.000	0.042
0.863	0.097	0.008	0.000
0.872	0.083	0.006	0.000
0.873	0.222	0.004	0.153
0.874	0.153	0.007	0.000
0.878	0.069	0.000	0.014
0.879	0.264	0.067	0.264
0.887	0.556	0.045	0.389
0.895	0.444	0.000	0.347
0.900	0.458	0.042	0.306
0.900	0.528	0.047	0.361
0.906	0.431	0.148	0.431

(b) Competitors

AUC	NF	NI	NNM
EBM			
0.902	1.000	0.008	1.00
Elastic-Net			
0.894	0.792	0.000	0.000
Random Forest			
0.839	1.000	1.000	1.000
XGBoost			
0.915	1.000	1.000	1.000

B DETAILS ON THE BENCHMARK EXPERIMENTS

We release all code for using *EAGGA* and replicating our results via https://github.com/slds-lmu/paper_2023_eagga. Benchmark experiments were run on an internal HPC cluster using Intel Xeon E5-2670 instances taking around 17700 CPU hours (benchmarks and ablation study). Total emissions are estimated to be an equivalent of roughly 1110 kg CO_2. Table 3 summarizes all tasks used in our benchmark experiments. Outer and inner resampling splits were fixed via different random seeds over the 10 replications but the same for all methods.

Table 3: OpenML tasks used in the benchmarks.

Task ID	Name	Number of Observations	Number of Features
37	diabetes	768	8
43	spambase	4601	57
3903	pc3	1563	37
3904	jm1	10885	21
3913	kc2	522	21
3918	pc1	1109	21
9946	wdbc	569	30
10093	banknote-authentication	1372	4
146819	climate-model-simulation-crashes	540	20
146820	wilt	4839	5
167120	numerai28.6	96320	21
168350	phoneme	5404	5
189922	gina	3153	970
190137	ozone-level-8hr	2534	72
190392	madeline	3140	259
190410	philippine	5832	308
359955	blood-transfusion-service-center	748	4
359962	kc1	2109	21
359972	sylvine	5124	20
359975	Satellite	5100	36

IDs correspond to OpenML task IDs, which enable querying task properties via https://www.openml.org/t/<id>. Task 3904 originally includes five observations with missing data, which were removed to allow for consistency over all tasks.

EAGGA was configured to use a population size of $\mu = 100$ and an offspring size of $\nu = 10$. The overall crossover probability was set to $p = 0.7$ and the overall mutation probability to $p = 0.3$. If crossover was to be applied, each hyperparameter of the search space of the learning algorithm underwent uniform crossover ($p = 0.5$) and crossover of group structures was performed as described in Section 4.1.2. If mutation was to be applied, each numeric and integer hyperparameter of the search space of the learning algorithm underwent Gaussian mutation ($p = 0.2, \sigma = 0.1$; values min-max scaled to $[0, 1]$ prior to mutation and re-transformed afterwards; values rounded to the closest integer in the case of integer hyperparameters), while each categorical hyperparameter underwent uniform mutation ($p = 0.2$), and each group structure was mutated by assigning each feature a new group membership with probability $p = 0.2$ and sampling a new monotonicity attribute for each group with probability $p = 0.2$. The hyperparameters of the initial population were constructed by using the default hyperparameters of the search space of the learning algorithm, which were then mutated as described above, except for one member of the population which was left unchanged. The group structures of the initial population were constructed using detectors as described in Section 4.2. We used the entropy-based information gain feature filter [36] as feature detector, a re-implementation of FAST [39] using a bin size

Table 4: EBM search space.

Hyperparameter	Type	Range	Trafo	Default
interactions	int.	$[0, \max(10, \lceil\sqrt{p(p-1)/2}\rceil)]$		10
outer_bags	int.	$[8, 50]$		8
inner_bags	int.	$[0, 50]$		0
max_rounds	int.	$\{5000, 10000\}$		5000
max_leaves	int.	$[2, 5]$		3
max_bins	int.	$[32, 1024]$	$\log_2$	256

"$\log_2$" in the Trafo column indicates that this parameter is optimized on a (continuous) logarithmic scale with base 2, i.e., the range is given by $[\log_2(\text{lower}), \log_2(\text{upper})]$, and values are re-transformed to the power of 2 prior to their evaluation. Parameters part of the full EBM search space that are not shown are set to their default. The Default column shows the values recommended by the EBM authors which were always used as the first initial design point.

Table 5: Elastic-Net search space.

Hyperparameter	Type	Range	Trafo
alpha	cont.	$[0, 1]$	
s	cont.	$[\exp(-7), \exp(7)]$	log

"log" in the Trafo column indicates that this parameter is optimized on a (continuous) logarithmic scale, i.e., the range is given by $[\log(\text{lower}), \log(\text{upper})]$, and values are re-transformed via the exponential function prior to their evaluation.

Table 6: XGBoost search space.

Hyperparameter	Type	Range	Trafo	Default
nrounds	int.	$[1, 5000]$	log	100
eta	cont.	$[1 \times 10^{-4}, 1]$	log	0.3
lambda	cont.	$[1 \times 10^{-4}, 1000]$	log	1
gamma	cont.	$[1 \times 10^{-4}, 7]$	log	1×10^{-4}
alpha	cont.	$[1 \times 10^{-4}, 1000]$	log	1×10^{-4}
subsample	cont.	$[0.1, 1]$		1
max_depth	int.	$[1, 20]$		6
min_child_weight	cont.	$[1, 150]$	log	$\exp(1)$
colsample_bytree	cont.	$[0.01, 1]$		1
colsample_bylevel	cont.	$[0.01, 1]$		1

"log" in the Trafo column indicates that this parameter is optimized on a (continuous) logarithmic scale, i.e., the range is given by $[\log(\text{lower}), \log(\text{upper})]$, and values are re-transformed via the exponential function prior to their evaluation. Parameters part of the full XGBoost search space that are not shown are set to their default. The Default column shows the initial values used as starting points for the initialization process in *EAGGA*.

of 10 as interaction detector and a monotonicity detector based on Spearman's ρ. Parents were selected via binary tournament selection using non-dominated sorting and crowding distance as criteria. Parents that resulted in zero features being selected where excluded from the tournament selection. A $(\mu + \nu)$ survival scheme was used based on non-dominated sorting and crowding distance.

In Section 5.1, we compare $EAGGA_{\text{XGBoost}}$ and $EAGGA_{\text{XGBoost}_{\text{md2}}}$ to an EBM, Elastic-Net, and XGBoost optimized for performance and an untuned random forest. Search spaces of the learning algorithms are given in Table 4, Table 5, and Table 6. The EBM, Elastic-Net, and XGBoost were optimized via sequential Bayesian Optimization similarly configured as SMAC [38], i.e., using a random forest as surrogate model and Expected Improvement [30] as acquisition function, which was optimized using a random search with a budget of 10000 function evaluations. The initial design of size $4d$ (d being the dimensionality of the search space) was sampled uniformly at random – except for the EBM, where the first initial design point was always given by the default configuration suggested by the EBM authors. $EAGGA_{\text{XGBoost}}$ and $EAGGA_{\text{XGBoost}_{\text{md2}}}$ also operate on the search space as given in Table 6, with the exception that `max_depth` was fixed to 2 for $EAGGA_{\text{XGBoost}_{\text{md2}}}$.

In some runs, evaluating the initial design during optimization of the EBM took longer than the whole compute budget of 8 hours (mostly for tasks 43, 10093, 189922, 190392, 167120, 190410, 168350, 359972 and 146820). In these cases, our fallback was to only evaluate the default configuration suggested by the EBM authors.

The random forest was implemented within XGBoost using `booster = "gbtree"`, `tree_method = "exact"`, `subsample = 1 - exp(-1)`, `colsample_bynode = 1 - exp(-1)`, `num_parallel_tree = 1000`, `nrounds = 1` and `eta = 1`.

In our analysis, we also inspected the best models found by each method and how they perform with respect to the AUC as well as NF, NI and NNM. Figure 5 visualizes the mean AUC of these best models. On the x-axis the average interpretability measures of these models are stated ($NF/NI/NNM$). We observe that black box models like XGBoost or random forests (RF) often rely on almost all features as well as many interactions of features when solely optimized for performance. The Elastic-Net models can be very sparse but often lack good performance. In contrast, the best models found by $EAGGA_{\text{XGBoost}}$ and $EAGGA_{\text{XGBoost}_{\text{md2}}}$ often perform almost on par with the XGBoost models solely optimized for performance but use substantially fewer features and interactions and often also rely on fewer non-monotone features. For example, on the philippine task, the best models found by $EAGGA_{\text{XGBoost}}$ result in an average AUC of around 0.860 using on average 29% of features, 11% of feature interactions and on average only 26% of the features have a non-monotone effect. In contrast, the XGBoost models optimized for performance result in an average AUC of around 0.864 but on average use all features, include interactions of all features and cannot guarantee that some of the features used in the model are restricted to have a monotone effect.

$\text{XGBoost}_{\text{MO}}$ in Section 5.2 was optimized via ParEGO using a random forest as surrogate model and Expected Improvement as acquisition function, which was optimized using a random search with a budget of 10000 function evaluations. The search space is given in Table 6. The initial design of size $4d$ was sampled uniformly at random.

We also computed the anytime dominated Hypervolume during optimization (i.e., calculated on the inner resampling), see Figure 6. Notably, the only tasks where $\text{XGBoost}_{\text{MO}}$ eventually outperforms $EAGGA_{\text{XGBoost}}$ are low-dimensional tasks with four or five features. A Wilcoxon signed-ranks test on the final mean dominated Hypervolume indicates that $EAGGA_{\text{XGBoost}}$ indeed solves the inner optimization problem better than $\text{XGBoost}_{\text{MO}}$ ($T = 30, p = 0.0026$).

C DETAILS ON THE ABLATION STUDY OF *EAGGA*

Here, we report detailed results of the ablation study of *EAGGA*. Figure 7 visualizes the anytime mean dominated Hypervolume during optimization (i.e., calculated on the inner resampling) of $EAGGA_{\text{XGBoost}}$ and different flavors as compared in our ablation study. Table 7 shows the corresponding final mean dominated Hypervolume. We observe that using detectors often results in a strong performance boost, but *EAGGA* without detectors usually catches up in performance. Generally, not performing either crossover or

mutation of group structures results in comparably poor performance, and performing neither crossover nor mutation results in final performance close to the random search.

We hypothesize that the effectiveness of using or not using crossover or mutation may depend on the performance of the detectors used to initialize the population in *EAGGA*. If the initial group structures determined by the detectors are already high-performing, then crossover during optimization could hinder progress due to excessive exploration. In such cases, using only mutation of the initial group structures may be more effective. On the other hand, if detector performance is poor, more exploration of group structures may be needed, and crossover during optimization can be beneficial. In conclusion, we believe that the performance of *EAGGA* can be further significantly enhanced by fine-tuning the configuration of mutation and crossover rates. Additionally, considering the possibility of changing these rates in a self-adaptive manner [37] could further improve the optimization process.

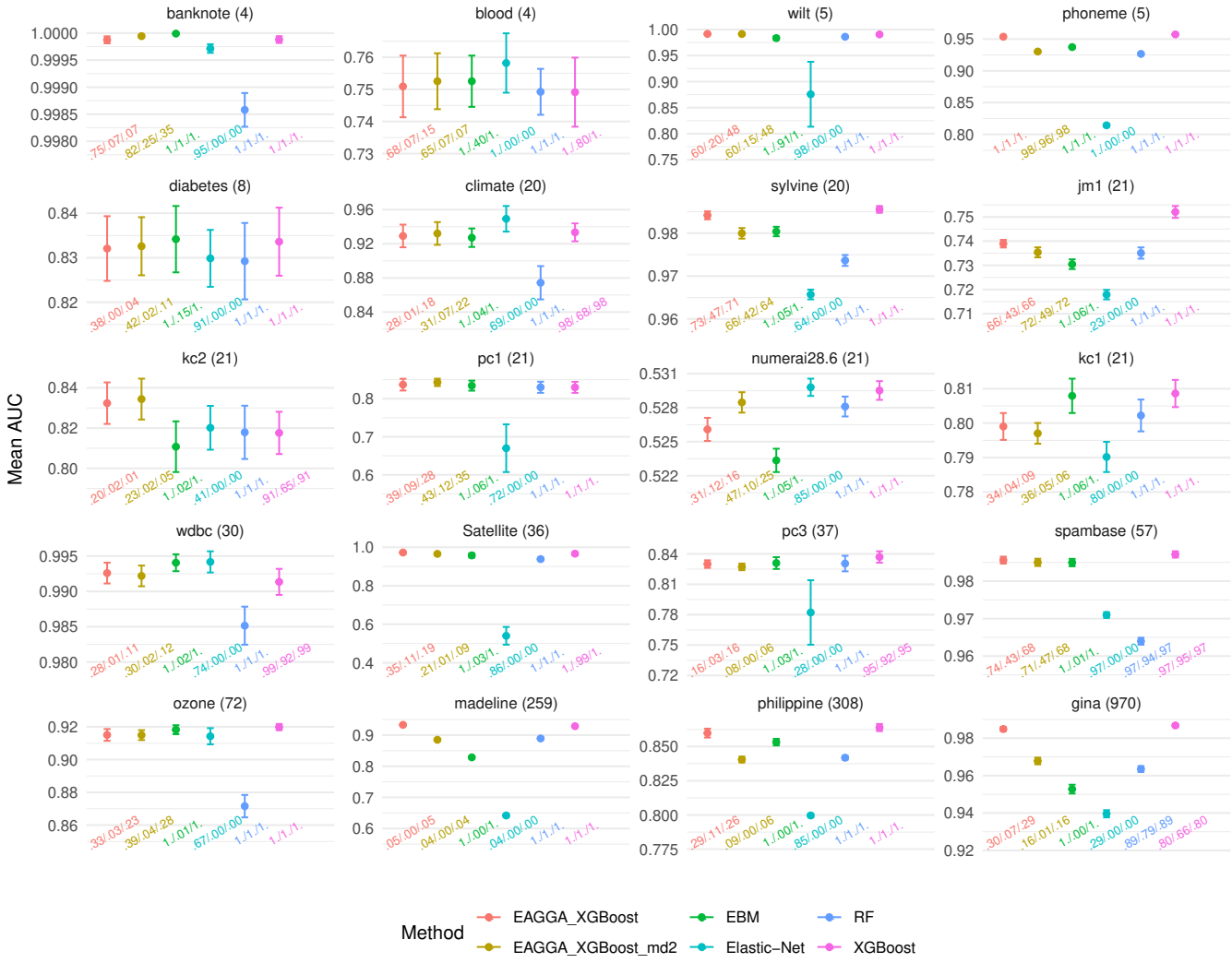

Figure 5: Mean AUC of the best models found by each method and their mean interpretability measures ($NF/NI/NNM$) averaged over 10 replications. Bars represent standard errors.

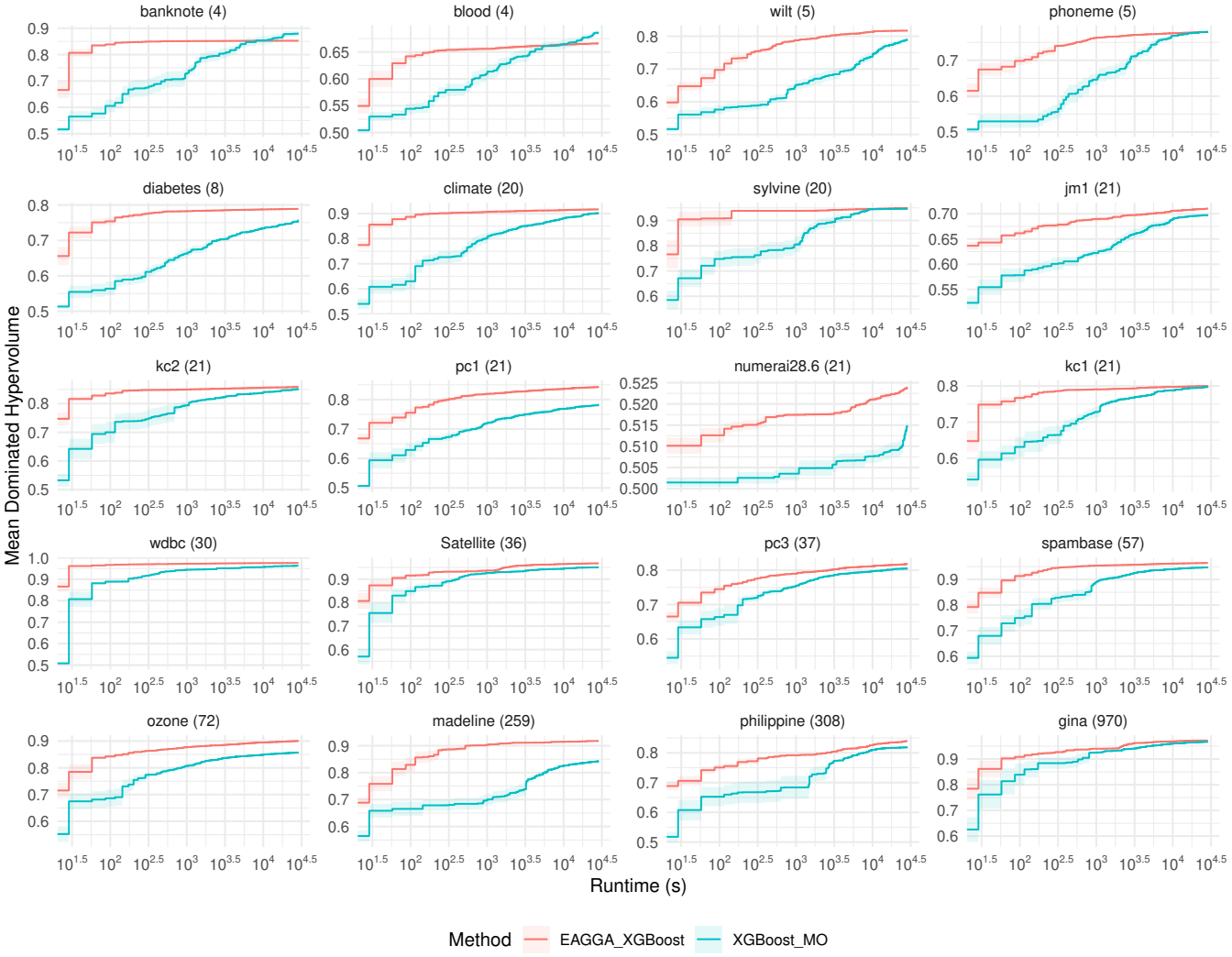

Figure 6: Anytime mean dominated Hypervolume during optimization of $EAGGA_{\text{XGBoost}}$ and $\text{XGBoost}_{\text{MO}}$ averaged over 10 replications. Ribbons represent standard errors. Note that the x-axis is shown on $\log_{10}$ scale.

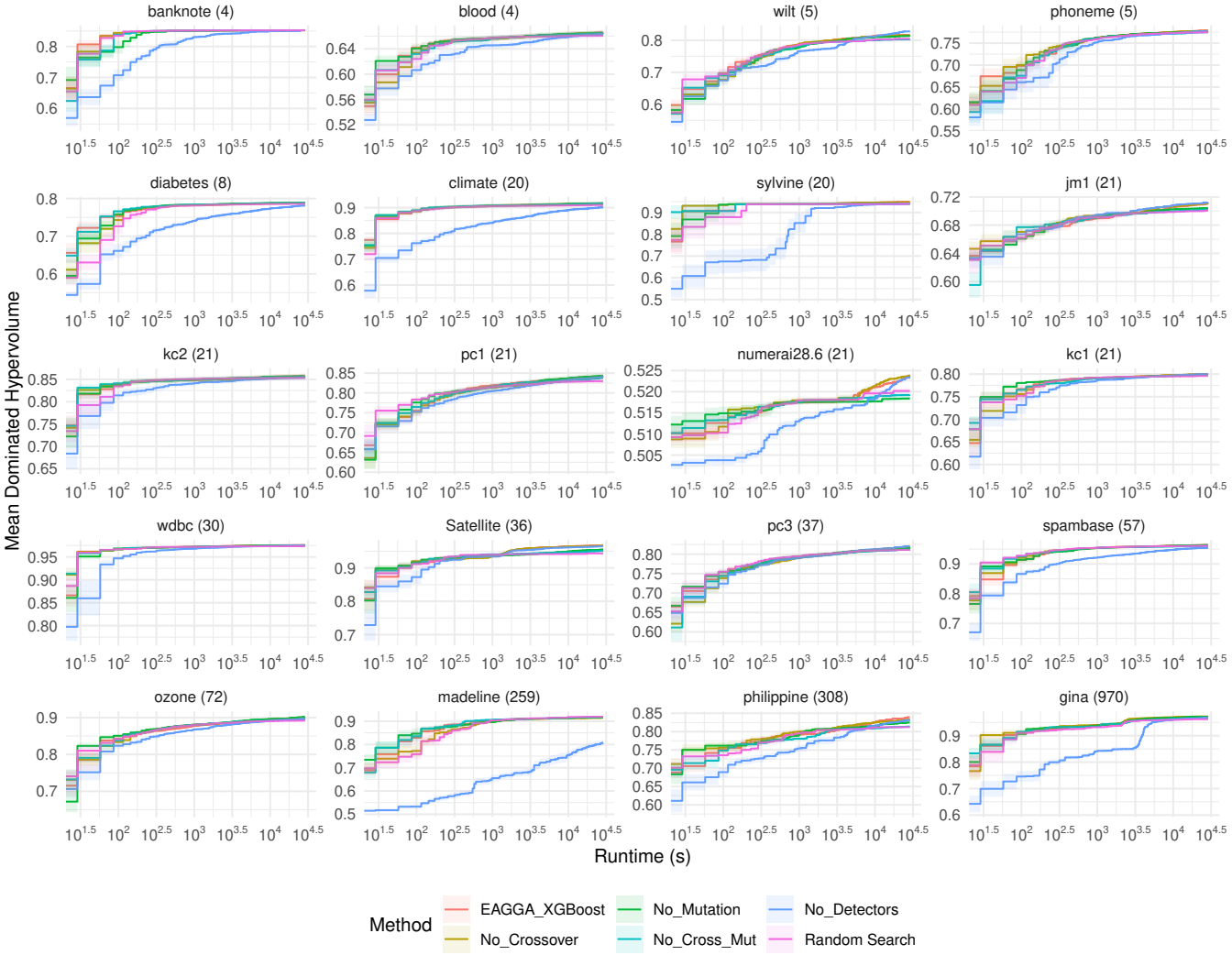

Figure 7: Anytime mean dominated Hypervolume during optimization of $EAGGA_{\text{XGBoost}}$ and different flavors averaged over 10 replications. Ribbons represent standard errors. Note that the x-axis is shown on $\log_{10}$ scale.

Table 7: Final mean dominated Hypervolume of $EAGGA_{\text{XGBoost}}$, different flavors and XGBoost_MO during optimization.

Task (p)	Method						
	EAGGA_XGBoost	No_Crossover	No_Mutation	No_Cross_Mut	No_Detectors	Random Search	XGBoost_MO
banknote (4)	0.853 (0.000)	0.852 (0.000)	0.852 (0.000)	0.853 (0.000)	0.852 (0.001)	0.852 (0.001)	**0.880** (0.012)
blood (4)	0.666 (0.004)	0.665 (0.004)	0.665 (0.004)	0.664 (0.004)	0.664 (0.004)	0.661 (0.004)	**0.686** (0.005)
wilt (5)	0.817 (0.001)	0.816 (0.002)	0.814 (0.003)	0.803 (0.003)	**0.828** (0.004)	0.805 (0.001)	0.789 (0.007)
phoneme (5)	0.779 (0.001)	0.778 (0.001)	0.777 (0.001)	0.775 (0.001)	0.777 (0.001)	0.775 (0.001)	**0.779** (0.004)
diabetes (8)	0.789 (0.003)	0.788 (0.003)	**0.789** (0.003)	0.789 (0.003)	0.782 (0.003)	0.786 (0.003)	0.757 (0.005)
climate (20)	0.917 (0.006)	0.916 (0.006)	**0.917** (0.006)	0.916 (0.006)	0.902 (0.008)	0.910 (0.007)	0.902 (0.007)
sylvine (20)	**0.949** (0.001)	0.947 (0.001)	0.940 (0.001)	0.940 (0.001)	0.941 (0.003)	0.939 (0.001)	0.946 (0.004)
jm1 (21)	0.710 (0.002)	0.710 (0.001)	0.704 (0.002)	0.702 (0.002)	**0.712** (0.001)	0.701 (0.002)	0.697 (0.002)
kc2 (21)	**0.858** (0.005)	0.855 (0.005)	0.857 (0.004)	0.855 (0.005)	0.855 (0.005)	0.853 (0.005)	0.851 (0.005)
pc1 (21)	0.843 (0.006)	0.839 (0.006)	**0.843** (0.007)	0.838 (0.006)	0.840 (0.006)	0.830 (0.005)	0.782 (0.006)
numerai28.6 (21)	**0.524** (0.001)	0.524 (0.000)	0.518 (0.001)	0.519 (0.001)	0.524 (0.001)	0.520 (0.001)	0.515 (0.001)
kc1 (21)	0.800 (0.002)	0.800 (0.002)	0.798 (0.002)	0.797 (0.002)	**0.800** (0.002)	0.796 (0.002)	0.797 (0.003)
wdbc (30)	**0.976** (0.001)	0.975 (0.001)	0.976 (0.000)	0.974 (0.001)	0.975 (0.001)	0.974 (0.001)	0.964 (0.003)
Satellite (36)	**0.968** (0.002)	0.967 (0.002)	0.955 (0.003)	0.951 (0.003)	0.965 (0.002)	0.944 (0.004)	0.950 (0.004)
pc3 (37)	0.818 (0.003)	0.816 (0.004)	0.817 (0.003)	0.812 (0.003)	**0.821** (0.004)	0.811 (0.004)	0.805 (0.004)
spambase (57)	**0.964** (0.001)	0.962 (0.000)	0.964 (0.001)	0.960 (0.000)	0.954 (0.002)	0.961 (0.001)	0.947 (0.001)
ozone (72)	0.901 (0.002)	0.899 (0.002)	**0.901** (0.002)	0.895 (0.002)	0.897 (0.002)	0.892 (0.003)	0.858 (0.003)
madeline (259)	0.918 (0.002)	0.915 (0.002)	0.918 (0.003)	0.918 (0.002)	0.807 (0.013)	**0.919** (0.002)	0.842 (0.004)
philippine (308)	**0.839** (0.003)	0.832 (0.003)	0.825 (0.003)	0.814 (0.002)	0.832 (0.003)	0.812 (0.002)	0.818 (0.001)
gina (970)	0.972 (0.001)	**0.972** (0.001)	0.972 (0.001)	0.966 (0.001)	0.967 (0.001)	0.962 (0.002)	0.967 (0.001)

Best final mean dominated Hypervolume highlighted in bold. Standard error over 10 replications in parentheses. "(0.000)" denotes that the standard error is smaller than 0.0005. For completeness we also include XGBoost_MO.