Master's Thesis

Gradually interpretable models via component-wise boosting

Author: Lorenz Haller

Supervisor: Prof. Dr. Bernd Bischl
Co-Supervisor: Christoph Molnar, M.Sc.
Department of Statistics
Ludwig–Maximilians–University Munich

Date: December 18, 2019

Abstract

An obstacle in model selection is the big gap between models that are highly interpretable and models that have high prediction accuracy. When selecting a model, often a choice is made between a simple model that can be interpreted well but has limited accuracy like a Linear Model or Decision Tree, and models that have high prediction accuracy but are hardly interpretable, e.g. Random Forests or Support Vector Machines. Therefore, there is an increasing demand in different application fields for models that address the issue of interpretability in addition to focusing on accuracy. This master thesis develops a framework in order to balance the trade-off between those two extremes.

The main idea is to incorporate component-wise boosting in a stage-wise manner. In each of the strictly separated stages, models from one specific category are boosted until no further improvement is made. For this purpose, a transition criterion is defined, which measures the risk decrease from one iteration to the next. If the improvement between two iterations in the same stage is not large enough any more, the algorithm passes on to the next stage, where a different kind of model is used.

In the first stage, simple linear models are used in order to explain linear effects. Afterwards, non-linear effects shall be taken into account as well, which is why splines or tree stumps are applied here. Beyond that, interactions of different degrees are considered by using trees in later stages.

The objective of this new method is to quantify how much each stage and model contribute to the predictions, on an average as well as on an individual level. The hereby developed reporting tool describes those contributions in different ways.

Finally, the approach is tested on a variety of data examples, which show its benefits as well as its limitations. In order to oppose the new approach to similar other methods, a benchmark study is performed, which targets on comparing the methods on a quantitative level in terms of accuracy as well as in terms of their interpretation abilities.

Contents

1 Introduction **1**
- 1.1 Relevance of Interpretability . . . 1
- 1.2 Trade-Off between Accuracy and Interpretability . . . 1
  - 1.2.1 Interpretable Models . . . 2
  - 1.2.2 Predictive Models . . . 4
  - 1.2.3 Post-hoc Interpretation . . . 5

2 Related Work **8**
- 2.1 GamBoost . . . 8
- 2.2 GA^2M . . . 8
- 2.3 Double Penalty Model for Interpretability . . . 9

3 Gradient Boosting and Component-wise Boosting **9**
- 3.1 Gradient Boosting . . . 9
- 3.2 Component-wise Boosting . . . 12

4 Gradually Interpretable Component-wise Boosting **13**
- 4.1 Requirements . . . 13
- 4.2 Input Parameters . . . 14
- 4.3 Main Method . . . 15
- 4.4 Prediction . . . 17
- 4.5 Hyperparameter Benchmark . . . 19
  - 4.5.1 Combination of ϵ and ν . . . 19
  - 4.5.2 Second Stage Model . . . 20

5 Visualization **21**
- 5.1 Interpretability Measures . . . 21
- 5.2 Data Example . . . 22
- 5.3 Global Interpretability . . . 22
  - 5.3.1 Partial Dependence Plots . . . 23
  - 5.3.2 ALE Main Effect Complexity . . . 24
  - 5.3.3 Risk Tables . . . 25
  - 5.3.4 Risk Plots . . . 27
- 5.4 Local Interpretability . . . 27
  - 5.4.1 Loss Tables . . . 27
  - 5.4.2 Barplots . . . 28

6 Benchmark **30**
- 6.1 Choice of Data Sets . . . 31
- 6.2 Comparison with other Methods . . . 32

7 Discussion **36**

List of Figures

1 Partial dependence plot for the estimated effect of the temperature 'Temp' on the predicted ozone measures. 6
2 Accumulated Local Effect plot for the estimated effect of the temperature 'Temp' on the predicted ozone measures. 7
3 A Feature Importance plot for the Airquality data set using a Random Forest. 7
4 Partial Dependence Plot of Solar.R for the Airquality data set using the Interpretable Component-wise Boosting Method. The black step function shows the estimated non-linear effect for the model using tree stumps, the red curve for using P-splines. . . . 24
5 ALE Plot of Solar.R for the Airquality data set. 25
6 Training Risk Table for the Airquality Data Set. 26
7 Test Risk Table for the Airquality Data Set. 26
8 Test Risk Table for the Airquality, Boston Housing and Bike Demand Data Sets. 27
9 Risk Curves and Feature Counter for the Airquality Data Sets. . 28
10 Individual Loss Table for the Airquality Data Set. 29
11 Absolute individual loss improvement per stage for the Airquality Data Set. 30
12 Prediction per stage for the Airquality Data Set. 30

List of Algorithms

1 Forward stagewise additive modeling 10
2 Gradient Boosting 11
3 Gradient Boosting using trees 11
4 Component-wise boosting 12
5 Interpretable component-wise boosting 17
6 Prediction Algorithm for Interpretable Component-wise Boosting 19

Notation

b_j^k	Base learner from base learner pool in stage k for feature j
c_j	Mean of corresponding terminal region R_j
$f^{[m]}(x)$	fitted function after iteration m
$\hat{f}(x)$	estimated prediction function
GAM	Generalized Additive Model
GLM	Generalized Linear Model
i	Observation index
j	Feature index $\in \{1, ..., J\}$
j*	Best selected feature in an iteration
k	Stage index $\in \{1, ..., 4\}$
L	Loss function
LM	Linear Model
m	Iteration $\in \{1, ..., M\}$
MMCE	Mean Misclassification Error
MSE	Mean Squared Error
PDP	Partial Dependence Plots
$Risk^{[m+1]}$	Risk after iteration m
$r^{[m](i)}$	Pseudo-residuals for observation i in iteration m
R_j	Terminal region in a decision tree
SVM	Support Vector Machine
t	t=$\{t^1, t^2, t^3, t^4\}$; transition iterations between stages 1-4
x	Feature vector
x_j	Feature j
xgboost	eXtreme Gradient Boosting
β	Coefficient vector (or step size in gradient descent)
ϵ	Minimum risk improvement rate and stage transition parameter
θ_j	Parameter vector (coefficient) of feature j for a base learner
$\hat{\theta}_j^{[m]}$	Estimated parameter vector (coefficient) of feature j for a base learner in iteration m
ν	Learning rate or shrinkage parameter
∇	Gradient

1 Introduction

1.1 Relevance of Interpretability

What is "interpretability"? Following Tim Miller's definition [28], "interpretability is the degree to which a human can understand the cause of a decision". Therefore, interpretability is closely connected to the human's potential of understanding the reasons for receiving a certain model outcome and prediction. In other words, the higher the interpretability, the easier it is for users to comprehend why a certain prediction is made.
To illustrate the relevance of interpretability in the field of data analysis, let's start with a motivating example.
A company authorizes a data science consulting company to help them increase their sales. They want to identify the customers, who are most likely to buy their product when contacted by phone. The consulting company then uses data from customers and previous calls to build a model for identifying the customers most likely to buy a product after being contacted by an employee. Assuming they use a Gradient Boosting or Random Forest model and get satisfying values for accuracy and AUC, the consulting company afterwards delivers a list of the most promising customers, who should be contacted for a sales call.
However, crucial for the success of these calls is not only which customer is selected, but also to incorporate the information for which reasons a specific person is contacted in the call. Those can be insights such as the position and decision-making power of a customer in his company as well as recent activity on the company's web page and certain product pages. In order to be able to deliver these data, the consulting company cannot use a model that focuses on accuracy only. It rather needs to additionally consider interpretability to find out which of a customer's features are essential for the prediction of the model and should be used to increase the call's probability of success.

This example was a first illustration why interpretability should be considered besides accuracy when working with machine learning methods. Even though interpretability recently is getting more into the focus in research and application, there is still a certain "gap" between prediction accuracy and interpretability. One often gets the impression that only the two extremes were possible: either using high performing methods like Random Forests but disregarding interpretability, or using worse performing methods like linear models which in exchange can be interpreted well.
The new approach in this thesis tries to minor this gap by combining both prediction accuracy and interpretability.

1.2 Trade-Off between Accuracy and Interpretability

Before I start explaining our new method, I want to cover different methods from statistical learning which play a role regarding the topic of interpretability. First, I will speak about interpretable models, which are models whose

results are already nicely interpretable themselves without the need of a further preparation. Here I will cover Linear Regression, Generalized Linear Models (GLMs), Generalized Additive Models (GAMs) and Decision Trees.
Secondly, I will speak about models that additionally need model-agnostic methods, in order to make them interpretable. In this section I will describe Random Forests and eXtreme Gradient Boosting. Afterwards, I will describe Partial Dependence Plots and Feature Importance Plots as examples of model-agnostic methods.
As this is a master thesis of limited extent, I will only cover the basics of the mentioned models and methods and talk about important aspects regarding the trade-off between accuracy and interpretability or later parts of this thesis. However, I won't go too much into detail regarding the functionality or implementation of each particular method as this would go beyond the scope.

1.2.1 Interpretable Models

The general problem definition is to estimate the outcome of a target y as good as possible using various features $x = \{x_1, ..., x_p\}$. For this purpose, different models have been developed.

The linear model is the most fundamental model in statistics [19, p. 11]. For a feature vector x, the model predicts the outcome of a target y using a model of the form $\hat{y} = \hat{\beta_0} + \sum_{j=1}^{p} x_j \hat{\beta_{j\cdot}} = x^T \hat{\beta}$.

For a linear regression model, the target is predicted as a weighted sum of the feature inputs. The estimated weights are often called coefficients. While there are many different methods of finding the optimal coefficients, the most common one is Least Squares, which tends to find those weights that minimize the squared differences between the actual and the estimated outcomes:

$$\hat{\beta} = \underset{\beta_0,...,\beta_p}{arg\,min} \sum_{i=1}^{n} (y^{(i)} - (\beta_0 + \sum_{j=1}^{p} \beta_j x_j^{(i)}))^2$$

Due to the linearity of the learned relationship the interpretation of a linear regression model is straightforward [29, p. 40-42]. For a numeric feature, increasing its value by one unit, while keeping all other features fixed, increases the estimated outcome by the coefficient. For binary categorical features, the coefficient of a feature is the estimated change of the outcome, when changing the particular feature from the reference category to the other category.

Due to its straightforward interpretations, the linear (regression) model is very popular in various fields of application. However, the model is only appropriate if the relationships between the features and the outcome are linear. With an increasing number of non-linear relationships and interactions the linear model loses prediction power.

Another problem for simple linear models is how they deal with large numbers of features. Having a linear model with hundreds of features that are used by the model makes interpreting difficult. A solution is to use sparse models with only few features. This can be achieved e.g. by using Lasso, which "performs feature selection and regularization of the selected feature weights" [29, p. 49-50].

A crucial problem for linear regression models is their dependence on different assumptions. If the target does not follow a Gaussian distribution or the relationship between features and the target is non-linear, linear regression models might not be a good choice any more.

For the first case of non-Gaussian outcomes, Generalized Linear Models (GLMs), which are able to model also other target types, are a more appropriate choice. For GLMs, the features are still modeled as a weighted sum, but are connected to the expected mean of the outcome distribution through a non-linear function g called link function: $g(E_Y(y|x)) = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p$. g can be chosen flexibly according to the data situation. For the outcome any distribution from the exponential family can be chosen, e.g. Binomial, Poisson or exponential. Also the Linear Regression and the Logistic Regression can be expressed as a GLM.
The interpretation of the coefficients of a GLM hence also depends on the assumed distribution and chosen link function. This works in a similar way than the interpretations for Linear Regressions, that's why going more into detail would extend the scope here. However, the usage of a link function, which is not the identity function, makes interpretations less straightforward [29, p. 67-77].

Secondly, if the relationship between the target and certain features is not linear, Generalized Additive Models (GAMs) are a widely used technique. GAMs were introduced by Hastie and Tibshirani [20] and extend Generalized Linear Models in the ability to learn non-linear relationships. GAMs replace the linear terms in the weighted sum of a GLM by more flexible functions, that don't have to be linear any more, but can be arbitrary functions of each feature: $g(E_Y(y|x) = \beta_0 + f_1(x_1) + ... + f_p(x_p)$. The f_j are linear combinations of infinite many basis functions B_1,B_2,... : $f_j(x) = \sum_{k=1}^{\infty} \beta_{jk} B_k(x)$. Estimating all the β_{jk} coefficients is one of the main tasks.
The most common way to learn non-linear relationships with GAMs are splines. A order-M spline is generally defined as a piece-wise polynomial of order M with continuous derivatives up to order $M - 2$ [19, p. 36]. Splines can then be combined in a way that various functions can be estimated. There are a lot of different types of splines and basis functions. For GAMs, P-splines have become a standard method for estimation, using a penalty based on higher-order differences of regression coefficients of equidistant knots [36]. In other words, P-splines penalise differences in the coeffcients between neighbouring B-spline basis functions [39]. Compared to other types of splines they are able to strongly reduce the computational effort.
The resulting curve is a linear combination of the spline weights, the feature

matrix and the knots, the interpretation of splines is done mainly visually by looking at the fitted curve for a feature.

GLMs and GAMs extend the classic linear models by loosening up their assumptions and giving back some flexibility to the estimations. This mostly also leads to an increase in accuracy, but as described earlier comes along with a little loss in interpretability. These methods therefore already start to address the trade-off between accuracy and interpretability that was mentioned at the beginning, but due to their inferiority in prediction performance compared to state-of-the-art machine learning methods, there is still room for improvement.

I now want to mention another quite popular method in the context of interpretability, the decision trees. The two best known algorithms for decision trees are the C4.5-algorithm from Quinlan [33] and the CART-algorithm from Breiman [6]. As the latter is more commonly used, I focus on that one. The functionality of CART is straightforward: a sample is repeatedly divided in subsamples, the nodes. Every node is split into to child nodes to the point of the terminal nodes. The partitioning is done using a certain split criterion on the features [37]. These criteria, also called impurity measures, can be the residual sum of squares for regression or the entropy or Gini impurity for classification. A split is always chosen for the particular feature and the particular split point which lead to the largest improvement of the impurity measure.
Big advantages of decision trees are their abilities to reflect non-linear relationships as well as interactions between features.
In addition, the interpretation of a decision tree is really intuitive. Starting in the root node, go down the tree, while for each split choosing the child node according to the value of the feature for the particular observation, until the terminal node is reached and a prediction is made. Decision trees are furthermore able to measure the feature importance for each feature in a data set by looking at the number of times a feature has been used and the reduction of impurity that was achieved by using the feature in a split.
The easy, user-friendly explanations are the main advantage of a decision tree. However, on the other hand, there are also some less beneficial properties, like their instability towards small changes in the data or their lack of smoothness [29, p. 83-84].

1.2.2 Predictive Models

In contrast to interpretable models, which can be interpreted on their own, there are also models that can only be interpreted post-hoc using model-agnostic methods.

Random Forests use bagging to combine different tree predictors to reduce variance and increase robustness [5]. In bagging, a model is applied to different bootstrap samples of the data set and the resulting estimates are averaged. Concretely, B samples of size N of the data are drawn with replacement. To reduce the correlation between the individual trees, only a certain number of

randomly chosen features are considered for each split. Apart from that, the trees are grown as described earlier. For regression the predictions of the individual trees are then averaged, for classification majority vote is used [19, p. 592].

The second model I want to mention here is eXtreme Gradient Boosting or XGBoost [11]. XGBoost is an implementation of Gradient Boosting using decision trees with strong performance and runtime. It has become very popular due to its good results for different competitions. I will go more into detail about Gradient Boosting in the next chapter, but wanted to mention XGBoost here because of its current leading role in machine learning. Both of the here explained models are typical examples for the application of post-hoc interpretation methods which will be discussed now.

1.2.3 Post-hoc Interpretation

I will present three model-agnostic methods, by name Partial Dependence Plots, Accumulated Local Effect Plots and Feature Importance Plots, and illustrate them by means of a data example.

Partial Dependence Plots (PDPs) show the marginal effect, e.g a linear effect, of one or two features on the predicted outcome of a particular model [16]. The partial dependence function for regression has the following form [29, p. 113]:

$$\hat{f}_{x_S}(x_S) = E(x_C)[\hat{f}(x_S, x_C)] = \int \hat{f}(x_S, x_C)\, d\,\mathbb{P}\, x_C$$

.

The vector x_S contains the features of interest, where want to have a look at their marginal effects. Then, "by marginalizing over the other features, one gets a function that depends only on features in S, interactions with other features included" [29, p. 113]. For a certain value of the features x_S, the partial dependence function then reflects the average predicted outcome.
The interpretation of PDPs therefore is quite intuitive. However, PDPs assume, that the features in x_S and in x_C are independent from each other and a violation of the assumption can cause unrealistic data points in the plot.

An alternative method which does not have this problem, are the so-called Accumulated Local Effect plots (ALE plots) that work with the conditional instead of the marginal distribution [1]. They calculate the differences in predictions for a particular grid of feature values. These differences are then interpreted as the effect the feature has for a particular instance.

The last model-agnostic method I want to cover in this section are feature importance plots. Feature importance is a measure for the relevance of a feature

for the prediction. In the case of decision trees and for a specific feature x_j, it can be calculated as sum of the squared improvements over all nodes where x_j was selected as the split variable [4]. This procedure naturally extends to Random Forests, only that now the importance for a feature x_j is accumulated over all individual trees [19, p. 593]. Another possibility of calculating the importance of features is the Permutation Feature Importance, which measures the increase in the prediction error of the model after the feature's values were permuted. The rationale is that if permuting the feature increases the error strongly, it is a good indicator that the feature is of importance for the prediction [29, p. 154].
The feature importance plots that are generated, show the most relevant features ranked by their feature importance.

For the purpose of illustration, I want to show a short example. I fitted a Random Forest for the Airquality data set in R using the `randomForest` package [26]. The target is the measure of ozone in ppb.
The following plots were generated using the `iml`-package [30]. The partial dependence plot (1) visualizes the effect of the temperature on the Fahrenheit scale on the ozone measure. Up to 76 degrees Fahrenheit, the temperature shows only a little effect on the predicted ozone. Then, between 76 and 90 degrees, increasing the temperature strongly increases the predicted ozone. Finally, if the temperature has reached 90 degrees, increasing it even more, has again only a little, even rather negative effect on the predicted outcome.
The ALE Plot 2 shows a similar trend for the effect of the temperature. Due to this likeness between the PDP and the ALE plot, one can conclude that the temperature is only marginally correlated to other features.

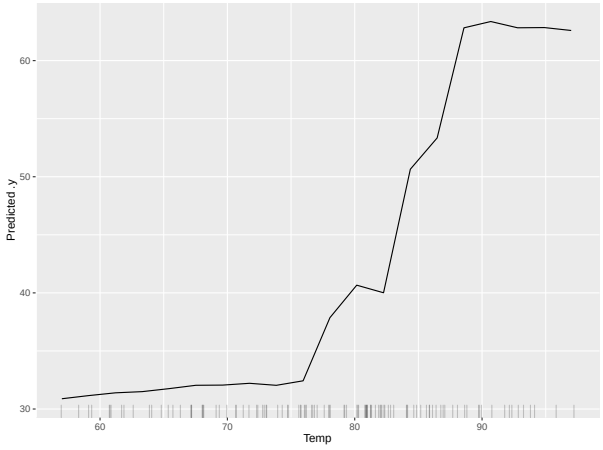

Figure 1: Partial dependence plot for the estimated effect of the temperature 'Temp' on the predicted ozone measures.

In addition, the Permutation Feature Importance is applied using the mean

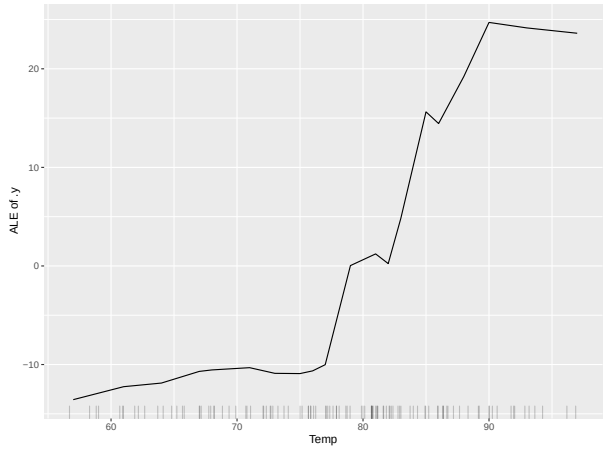

Figure 2: Accumulated Local Effect plot for the estimated effect of the temperature 'Temp' on the predicted ozone measures.

squared error as loss measure for the performance decrease. The Feature Importance Plot (3) shows that the most important feature for the prediction of ozone is the temperature (Temp), followed by the wind (Wind) and the solar radiation (Solar.R).

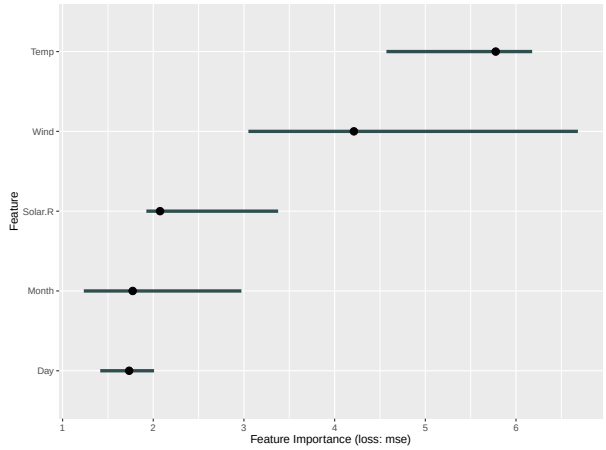

Figure 3: A Feature Importance plot for the Airquality data set using a Random Forest.

There are also a lot more model-agnostic methods, which I cannot cover in more detail. The most popular ones are Individual Conditional Expectation (ICE) plots, Global surrogate models, Local surrogate model (especially using

the LIME framework) and Shapley Values. For more information, I refer to [29].

2 Related Work

In this chapter I will cover methods that explicitly try to solve the gap between accuracy and interpretability. These methods try to achieve decent performance while still being highly interpretable. Here I will take a look at GamBoost as well as GA^2M.

2.1 GamBoost

I have already introduced GAMs in the previous chapter. Tutz and Binder [39] came up with a new method for the estimations of the β-coefficients called GAMBoost in 2006. They use a component-wise boosting approach where only one feature is fitted in each iteration. For boosting I will go more into detail later. However, the baselearners they use in their implementation in the R package GAMBoost are P-splines with 20 evenly spaced knots, with a penalty based on second order differences using a smoothing parameter λ [39]. As stopping criteria they consider the AIC and cross-validation. The resulting model is still a GAM, thus it can be interpreted as explained earlier.

2.2 GA^2M

Lou et al. (2013) [27] address the problem, that GAMs show significantly lower accuracy compared to more complex models. They solve this issue by adding selected terms of pairwise interactions to the standard GAMs. The authors therefor propose a new method for smarter exploration of feature pairs called FAST, which efficiently ranks all possible pairs.
FAST starts with the best additive model and then tries to detect interactions on the residuals. To measure the possible benefit of modeling a certain interaction by $f_{ij}(x_i, x_j)$, they look at the residual sum of squares (RSS) of the interaction model f_{ij} using the idea that if a strong interaction (x_i, x_j) exists, modeling f_{ij} significantly reduces the RSS.
Using the FAST method, the authors are able to rank all possible feature pairs and add the best interaction to the model. Then, in a first stage they build the best additive model using only one-dimensional components. For the second stage, all one-dimensional components are fixed and pairwise interaction models are built on the residuals. However, they don't use P-splines as the authors of GAMBoost propose, but gradient boosting with shallow tree-like ensembles.
A fast implementation of the GA^2M model is the Explainable Boosting Machine (EBM) from the open source Python package InterpretML [32]. By plotting the individual f_j functions, the feature contributions to the predictions get understandable. At the same time, the accuracy is improved by using techniques

like Gradient Boosting or Bagging, which makes EBM comparable to state-of-the-art methods [32].

2.3 Double Penalty Model for Interpretability

Wang and Zhou [40] propose a model that includes two penalty terms to balance the trade-off between the interpretable and the non-interpretable part of a prediction model. In their opinion, each model can be divided into interpretable and non-interpretable function parts. They came up with two function classes, one class $\mathscr{F}$ for models that are easily interpretable and the other $\mathscr{G}$ for models that are assessed as "uninterpretable". Their goal is to find the best function from each class. For estimating they propose a double penalty model:

$$(\hat{f}, \hat{g}) = \underset{f \in \mathscr{F},\, g \in \mathscr{G}}{arg\,min} \; \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i) - g(x_i))^2 + L_f(f) + L_g(g)$$

For that matter L_f and L_g are convex penalty functions on f and g. As solution to this optimization problem they suggest an iterative procedure for solving two separate problems similar to coordinate descent.
They look at the cases where the two function classes are separable and where they are not and claim that in the separable case, the proposed algorithm converges very fast.
For instance, they use the Lasso algorithm for the interpretable and an xgboost model for the non-interpretable part. Therefor an L_1 penalty is used for Lasso and an L_2 penalty is used for xgboost. For both models they are able show their correlation with the target, depending on the strength of the penalization. In addition, they explain how to maximize the predictive power of the interpretable part of the algorithm while keeping the overall accuracy almost equal.

3 Gradient Boosting and Component-wise Boosting

As our method is embedded in the framework of component-wise boosting, gradient boosting and more detailed component-wise boosting are explained now.

The general idea of boosting is to combine several weak learners in such a way that the resulting learner is a strong learner with a better performance. Weak learners are learners, that are slightly better than guessing, also known as base learners.

3.1 Gradient Boosting

The gradient boosting algorithm is based on the gradient descent technique from numerical optimization [16].
The negative gradient $-\nabla f(x)$ is the direction of the steepest descent. From

a starting point $x^{[k]}$, the next step in minimization is given by $x^{[k+1]} = x^{[k]} - \nu \nabla f(x^{[k]})$. ν is called step size or "shrinkage parameter" and controls the learning rate of the procedure. It can be set through different ways, e.g. line search [16].

In a regression problem for example, where one wants to learn an additive model in the form of $f(x) = \sum_{m=1}^{M} \beta^{[m]} b(x, \theta^{[m]})$, the aim is to minimize the empirical risk $R_{emp}(f) = \sum_{i=1}^{n} L(y^{(i)}, f(x^{(i)})) = \sum_{i=1}^{n} L(y^{(i)}, \sum_{m=1}^{M} \beta^{[m]} b(x, \theta^{[m]})$. L here is an arbitrary loss function, e.g. the squared loss.

As it is difficult to minimize all $\beta^{[m]}, \theta^{[m]}$ simultaneously, additive components are added in a greedy fashion by minimizing the risk w.r.t. the next component. This iterative procedure is called forward stagewise additive modeling [19, p. 342]. The algorithm is defined as follows:

1. Initialize $\hat{f}^{[0]}(x)$;
2. **for** $m = 1 \rightarrow M$ **do**
   - $\hat{\beta}^{[m]}, \hat{\theta}^{[m]} = \underset{\beta, \theta}{arg\,min} \sum_{i=1}^{n} L(y^{(i)}, \hat{f}^{[m-1]}(x^{(i)}) + \beta\, b(x^{(i)}, \theta))$;
   - Update $\hat{f}^{[m]}(x) \leftarrow \hat{f}^{[m-1]}(x^{(i)}) + \hat{\beta}^{[m]}\, b(x, \hat{\theta}^{[m]})$

end

Algorithm 1: Forward stagewise additive modeling

In each iteration, the algorithm aims to find the new additive component $b(x, \theta^{[m]})$ and corresponding weight coefficient $\beta^{[m]}$ in each iteration m, which can be done using gradient descent. This is done by calculating the derivative $\frac{\partial R_{emp}}{\partial f(x^{(j)}}$ with respect to each component of the parameter vector. As result a gradient descent update has the form $f(x) \leftarrow f(x) - \beta \frac{\partial L(y, f(x))}{\partial f(x)}$. The last part, which determines the direction of the gradient descent is called "pseudo residuals", which are more precisely defined as $r^{[m](i)} = -[\frac{\partial L(y^{(i)}, f(x^{(i)}))}{\partial f(x^{(i)})}]_{f=f^{[m-1]}}$. For the squared loss the pseudo residuals match the normal residuals.
The gradient boosting algorithm has the following form [19, p. 361]:

1. Initialize $\hat{f}^{[0]}(x) = \underset{\theta}{arg\,min} \sum_{i=1}^{n} L(y^{(i)}, \theta)$;
2. **for** $m = 1 \to M$ **do**
   For all i: $r^{[m](i)} = -[\frac{\partial L(y^{(i)}, f(x^{(i)}))}{\partial f(x^{(i)})}]_{f=f^{[m-1]}}$;
   Fit a regression learner to the pseudo residuals $r^{[m](i)}$:
   $\hat{\theta}^{[m]} = \underset{\theta}{arg\,min} \sum_{i=1}^{n} (r^{[m](i)} - b(x^{(i)}, \theta))^2$;
   Line search $\hat{\beta}^{[m]} = \underset{\beta}{arg\,min} \sum_{i=1}^{n} L(y^{(i)}, f^{[m-1}(x) + \beta b(x, \theta^{[m]}))$;
   Update $\hat{f}^{[m]}(x) = \hat{f}^{[m-1]}(x) + \hat{\beta}^{[m]}\, b(x, \hat{\theta}^{[m]})$;
end
3. Output: $\hat{f}(x) = f^{[M]}(x)$

Algorithm 2: Gradient Boosting

One boosting iteration is exactly one approximated gradient step, which minimizes the empirical risk as much as possible.

The most commonly used base learners for Gradient Boosting are trees, which can also simply be written in an additive structure: $b(x; \{R_j, c_j\}_1^J) = \sum_{j=1}^{J} c_j \mathbb{I}(x \in R_j)$ [19, p. 356]. For the algorithm this results in:

$$f^{[m]}(x) = f^{[m-1]}(x) + \beta^{[m]} \sum_{j=1}^{J^{[m]}} c_j^{[m]} \mathbb{I}(x \in R_j^{[m]})$$

The gradient boosting algorithm using trees as base learners then looks as follows:

1. Initialize $\hat{f}^{[0]}(x) = \underset{\theta}{arg\,min} \sum_{i=1}^{n} L(y^{(i)}, \theta)$;
2. **for** $m = 1 \to M$ **do**
   For all i: $r^{[m](i)} = -[\frac{\partial L(y^{(i)}, f(x^{(i)}))}{\partial f(x^{(i)})}]_{f=f^{[m-1]}}$;
   Fit a regr. tree to the pseudo residuals $r^{[m](i)}$ giving the terminal regions $R_j^{[m]}$;
   for $j = 1 \to J^{[m]}$ **do**
      $\hat{c}_j^{[m]} = \underset{c}{arg\,min} \sum_{x^{(i)} \in R_j^{[m]}} L(y^{(i)}, f^{[m-1]}(x^{(i)} + c)$
   end
   End for $\hat{b}^{[m]}(x) = \sum_{j=1}^{J} \hat{c}_j \mathbb{I}(x \in R_j)$;
   Update $\hat{f}^{[m]}(x) = \hat{f}^{[m-1]}(x) + \hat{b}^{[m]}(x)$;
end
3. Output: $\hat{f}(x) = f^{[M]}(x)$

Algorithm 3: Gradient Boosting using trees

In order to avoid overfitting, either the number of boosting iterations M or the depth of the used trees may be limited. A shrinkage parameter ν is used to control the learning rate [19, p. 364]. The update step of the algorithm then has the form $\hat{f}^{[m]}(x) = \hat{f}^{[m-1]}(x) + \nu\, \hat{\beta}^{[m]}\, b(x, \hat{\theta}^{[m]})$. The best choice for ν seem to be values smaller than 0.1 [19, p. 365].

3.2 Component-wise Boosting

Gradient boosting, especially when using trees as base learners, has huge predictive power, but is difficult to interpret unless only tree stumps are used. Component-wise boosting [8] wants to find a model with high predictive performance which still has interpretable components. Therefore it tries to use interpretable base learners, which lead to known statistical models, which is why it is also called "model-based boosting" or "statistical boosting".
The main difference compared to the classical gradient boosting is that no longer only one type of base learner is used, but a set of base learners. This set is defined by

$$b_j^{[m]}(x, \theta^{[m]})$$

with $j = 1, ...J$, where j defines the type of base learner. In each iteration, the best base learner is chosen. Often base learners are not defined on the whole feature vector x , but only on one specific feature x_j, so that

$$b_j^{[m]}(x_j, \theta^{[m]})$$

with $j = 1, ..., p$. Additionally, the base learners are restricted to additive models. The algorithm for component-wise boosting has the following form:

1. Initialize $\hat{f}^{[0]}(x) = \underset{\theta}{arg\,min} \sum_{i=1}^{n} L(y^{(i)}, \theta)$ 2. **for** $m{=}1 \rightarrow M$ **do**

   For all i: $r^{[m](i)} = -[\frac{\partial L(y^{(i)}, f(x^{(i)})}{\partial f(x^{(i)})}]_{f=f^{[m-1]}}$;
   For all $j = 1, ..., J$: Fit a regr. learner b_j to the pseudo residuals $r^{[m](i)}$: $\hat{\theta}_h^{[m]} = \underset{\theta_j}{arg\,min} \sum_{j=1}^{n} (r^{[m](i)} - b_j(x^{(i)}, \theta_j))^2$;
   End for $j^* = \underset{j}{arg\,min} \sum_{j^=1}^{n} (r^{[m](i)} - b_j(x^{(i)}, \theta_j^{[m]}))^2$;
   Update $f^{[m]}(x) = f^{[m-1]}(x) + \nu b_{j^*}(x, \theta_{j^*}^{[m]})$;

   end
3. End for output $\hat{f}(x) = f^{[M]}(x)$

Algorithm 4: Component-wise boosting

An advantage is, that one can fit to various distributions and loss functions as long as the derivative of the loss can be calculated. This means one can use linear base learners and make the procedure equivalent to fitting linear models or GLMs. Or one uses non-linear base learners like P-splines and the procedure

will be equivalent to fitting GAMs. Performance-wise component-wise boosting however will usually be inferior to gradient boosting with trees.

4 Gradually Interpretable Component-wise Boosting

The approach that is introduced in this section is a multi-stage method using component-wise boosting implemented in R [34]. The method has four strictly separated stages. In each stage, a different base learner is fitted to the data via a component-wise boosting procedure using a wrapper for the `mboost` package [24]. Later stages fit models to the residuals from previous learners. As soon as a stage cannot improve the accuracy any more, the method passes on to the next stage.

The following table presents an overview over the different stages:

Stage 1:	Linear components	Linear regression
Stage 2:	Non-linear components	Splines or Tree Stumps
Stage 3:	Two-dimensional components	Trees of depth = 2
Stage 4:	More-dimensional components	Trees of depth > 2

4.1 Requirements

This method tries to make an appropriate trade-off between accuracy and interpretability. A crucial line of thought was the following: what would the ideal method look like that considers both, accuracy and interpretability in a reasonable way?
In terms of accuracy, the method should be able to reach similar or only slightly worse results than state-of-the-art methods which mainly focus on the aspect of accuracy.

One point worth mentioning is the duration in terms of the number of iterations, that a boosting method needs in order to reach a certain level of accuracy. It is apparent, that consecutively boosting a linear model, a spline model and two tree models, each fitted to the particular residuals, takes more iterations to reach the same accuracy as a model that boosts deeper trees right from the start. Therefore, run-time in terms of iterations is not a focus.
However, the method considers the number of iterations via a transition parameter ϵ to make sure the algorithm passes on the next stage as soon as the current stage is not improving enough in terms of accuracy any more. In essence, this method is not trying to compete with other methods regarding the number of iterations needed to reach a certain accuracy level, but still takes this dimension into account aiming to reach comparable numbers.

When it comes to interpretability, the method should answer the following questions:

- What is an adequate manner of modeling the relationship between the target and the features? Is it enough to model the relationship linearly or are non-linear models needed? Should interactions between features be included and if so, of which order?
- How does a particular feature influence the prediction?
- Which features are the most important ones and have the largest influence on the prediction?
- How does the number of used features influence the accuracy?
- Can statements about single observations be made? How does a single prediction come about?

For arbitrary complex data sets, the method should be able to recognize and point out the level of complexity, apply appropriate models and illustrate the results in a reasonable way. For example, if the influence of a certain feature on the prediction is completely linear, the method should be able to identify that and subsequently illustrate the relationship through a linear effect.

In addition, the method should indicate how much accuracy is gained by adding a more complex stage and tell if a linear approach or a non-linear approach without interactions are sufficient for a specific task. That is why the method should supply an overview for the different stages, which shows their particular accuracy improvements.

4.2 Input Parameters

To get a first overview, here's how the function is defined in R:

```
icb <- function(data, formula, target_class="Gaussian",
                nu=0.1, epsilon = 0.001,
                bl2=c("bbs","btree"),
                max_depth = 8, min_split = 20,
                min_bucket = round(min_split/3))
```

First, the method requires `data` in the form of `data.frame` and a `formula` that defines the target variable as well as the features that should be included in the model. Two possible classes for the target variable `target_class` are implemented: `Gaussian` for Regression, and `Binomial` for Classification.

The method uses the loss and gradient implementations from mboost [24]. For regression, the loss function is the squared error $L(y, f) = (y - f)^2$ with the corresponding gradient being $y - f$ (see [21]).
For Classification, the loss function is the negative binomial log-likelihood loss

$$L(y, f) = -[y \log(\pi(f)) + (1 - y) \log(1 - \pi(f))] = \log(1 + \exp(-2\tilde{y}f))$$

, where $\tilde{y} = 2y - 1$ and $\pi(f) = \mathbb{P}(y = 1 \,|\, x)$. In this context the term risk is defined as the weighted sum of the individual losses: $\sum w * L(y, f)$. As in our case all weights are equal, the risk is simply a sum of the individual losses.

Furthermore, the parameter ν (`nu`) defines the step size or shrinkage parameter or learning rate. As already introduced in section 3, it controls how fast the algorithm is learning. It restricts how much the overall model includes from the model in one particular iteration.

The ϵ (`epsilon`) parameter controls the transition from one stage to the next as well as the stopping point of the method. The ϵ represents the minimum necessary improvement rate of the risk from one iteration to the next. That means it is a relative value which is used to decide whether the improvement of the accuracy, i.e. the reduction of the risk, from one iteration to another is still large enough to stay in the current stage. If not, the method passes on to the next stage. If the improvement is not large enough any more at one point in the last stage, the model fitting will end.

The `bl2` parameter decides which model is used in the second stage: tree stumps (choose `bl2` = "`btree`") or p-splines (choose `bl2` = "`bbs`").

The final three parameters are crucial for the stages where trees are fitted: The `max_depth` parameter defines the depth of the trees in the last stage. The `min_split` parameter describes the "minimum sum of weights in a node in order to be considered for splitting" [23], which in our case using equal weights is equivalent to the minimum number of observations that needs to be in a node in order for considered for a split.
The other parameter, `min_bucket`, is, again using the same assumption of equal weights, defined as the minimal number of observations in a terminal node.

As the choice of all the parameters strongly depends on the data, hyperparameter tuning is recommended.

4.3 Main Method

Before I start with the description of the method, I want to have a look at the preparation steps in the method that are done before. For binary classification tasks, the target is re-encoded to $\{-1, 1\}$ to make it consistent with the internal encoding of mboost. As described for the input parameters, then the risk and gradient function are extracted from mboost depending on the `target_class` input parameter.
The model fitting starts afterwards by fitting an intercept model. This is done using `lm.fit` for regression and `glm.fit` with the logit-link for classification.

Then the algorithm starts with the first stage, where linear models are fitted. Therefore, first an mboost model with one iteration (`mstop` = 1) is fitted using `baselearner` = "`bols`" to fit linear models. As an offset the fitted values from the intercept model are used. Afterwards, a while-loop is used to add an additional mboost iteration in every iteration while at the same time making sure that the risk reduction from one iteration to the next relatively is at least ϵ.

The number of iterations after the first stage is saved as it is used in the next stage, where non-linear components are added, either tree stumps or P-splines with a B-spline basis as described in Schmid and Hothorn (2008) [36]. As mentioned before, the decision which method is used in the second stage, is made by the user via the input parameter `bl2`. Similar to the first stage an mboost model is fitted to the data, using the fitted values after the first stage as an offset. Again, the mboost model is trained forward afterwards, adding one component to the model per iteration until the achieved risk reduction is not large enough any more.

As before, the iteration number after the second stage is saved and the algorithm goes over to the third stage, where conditional inference trees (see [22]) of depth two (setting `maxdepth` = 2) are fitted and the goal is to detect the best two-dimensional interaction in each iteration. Here the `blackboost` function from mboost is applied using the `ctree_control` parameter from the `partykit` package [23] to specify the hyperparameters `minsplit`, `minbucket` and `maxdepth` for the conditional inference trees. The further iterations of the stage are then performed in a similar way as in the first two stages.

The fourth stage works similarly as the third stage with the difference that now deeper trees with a depth specified by `max_depth` are fitted.

As a result, the method returns a list containing the fitted values, the numbers of the transition iterations, the risk from each iteration, the fitted mboost models from all four stages, the input parameters as well the data and feature names and the risk function.

In summary, our new method can be described by the following algorithm:

Data: data, formula, ν, ϵ, target_class, bl2, df_spline, max_depth, min_split, min_bucket

1. Initialize $\hat{f}^{[0]}(x) = \underset{\theta}{arg\,min} \sum_{i=1}^{n} L(y^{(i)}, \theta)$;

$Risk^{[1]} = Risk(y, \hat{f}^{[0]}(x))$;

Iteration m = 0 ;

Offset = $\hat{f}^{[0]}(x)\ \forall\, i \in \{1, ..., n\}$;

2. **for** *stage k=1* $\rightarrow$ *4* **do**

Iteration m = m+1 ;

For all i: $r^{[m](i)} = -[\frac{\partial L(y^{(i)}, f(x^{(i)})}{\partial f(x^{(i)})}]_{f=\hat{f}^{[m-1]}}$;

For all $j = 1, ..., J_k$: Fit a stage learner b_j^k to the pseudo residuals $r^{[m](i)}$ using the offset: $\hat{\theta}_j^{[m]} = \underset{\theta_j}{arg\,min} \sum_{j=1}^{n} (r^{[m](i)} - b_j^k(x^{(i)}, \theta_j))^2$;

End for $j^* = \underset{j}{arg\,min} \sum_{j^=1}^{n} (r^{[m](i)} - b_j^k(x^{(i)}, \hat{\theta}_j^{[m]}))^2$;

Update $\hat{f}^{[m]}(x) = \hat{f}^{[m-1]}(x) + \nu\, b_{j^*}^k(x, \hat{\theta}_{j^*}^{[m]})$;

$Risk^{[m+1]} = Risk(y, \hat{f}^{[m]}(x))$;

while ($\frac{Risk^{[m]}}{Risk^{[m+1]}} \geq (1 + \epsilon)$) **do**

Iteration m = m+1 ;

For all i: $r^{[m](i)} = -[\frac{\partial L(y^{(i)}, f(x^{(i)})}{\partial f(x^{(i)})}]_{f=\hat{f}^{[m-1]}}$;

For all $j = 1, ..., J_k$: Fit a stage learner b_j^k to the pseudo residuals $r^{[m](i)}$ using the offset: $\hat{\theta}_j^{[m]} = \underset{\theta_j}{arg\,min} \sum_{j=1}^{n} (r^{[m](i)} - b_j^k(x^{(i)}, \theta_j))^2$;

End for $j^* = \underset{j}{arg\,min} \sum_{j^=1}^{n} (r^{[m](i)} - b_j^k(x^{(i)}, \hat{\theta}_j^{[m]}))^2$;

Update $\hat{f}^{[m]}(x) = \hat{f}^{[m-1]}(x) + \nu\, b_{j^*}^k(x, \hat{\theta}_{j^*}^{[m]})$;

$Risk^{[m+1]} = Risk(y, \hat{f}^{[m]}(x))$;

end

Update offset[i] = $\hat{f}^{[m]}(x_i)$;

Pass on to next stage or end algorithm after stage 4

end

3. End for output $\hat{f}(x) = \hat{f}^{[M]}(x)$

Algorithm 5: Interpretable component-wise boosting

4.4 Prediction

The prediction function of our method takes three input parameters:

```
icb_predict <- function(icb_object, newdata, target)
```

The input needs to contain a trained `icb_object` received from the main method, the new data (`newdata`) that will be predicted and optionally a `target` parameter, where the name of the target in the new data can again be specified in order to not only create predictions for the new data but at the same time calculating the risk in every iteration. So, if `target` is filled, one can calculate the risk in every iteration, using the risk function from `icb_object`, the true target values and the predictions in every iteration.

For the first iteration, the prediction for every observation is set to the value received from the intercept model of the training object. Then, for each iteration the particular `mboost` object is used to make predictions.

However, to get the predictions after the first stage, one has to add the predictions from the intercept model and from the last iteration of the first stage. This results from using an offset in the method, so that in every stage and in each iteration the model improves based on the fitted values from the previous stage.

For the later stages, the prediction procedure works the same way. To get the predictions for a particular stage, the predictions from a previous stage have to be added to the final predictions from this stage. In summary, the final predictions result from adding the intercept model prediction and the predictions from the individual stages.

The functionality of the prediction method can be visualized by the following algorithm:

Data: icb_object, newdata, target

t = $\{t^1, t^2, t^3, t^4\}$ = TransitionIteration$_{icb_object}$;
Iteration m = 0 ;
Prediction = $\hat{f}^{[0]}_{icb\_object}$(newdata) ;
if *!is.null(target)* **then**
  Risk$^{[m+1]}$ = Risk(y, Prediction)
end

for *stage k=1 → 4* **do**
  $\hat{f}^k = \hat{f}^k_{icb\_object}$;
  while $m \leq t^k$ **do**
    Iteration m = m+1 ;
    Pred_Iteration$^{[m]}$ = $\hat{f}_{k[m]}$(newdata) ;
    if *!is.null(target)* **then**
      Risk$^{[m+1]}$ = Risk(y, Prediction + Pred_Iteration)
    end
  end
  Prediction = Prediction + Pred_Iteration$^{[t^k]}$;
end

Output: Prediction

Algorithm 6: Prediction Algorithm for Interpretable Component-wise Boosting

4.5 Hyperparameter Benchmark

4.5.1 Combination of ϵ and ν

When applying this new method, ϵ and ν are quite crucial hyperparameters . ν, the shrinkage parameter or step size, determines the model's learning rate. ϵ, the minimum risk improvement rate, replaces the `mstop`-parameter from original boosting implementations as stop criterion. But furthermore it serves as indicator when to pass on to the next stage.
The choice of the default values in our method for ϵ and ν were determined by a benchmark study of seven different data sets - Airquality [10], Boston Housing [18], Wine [14], kin8nm [13], Spam [15], Pima Indians Diabetes [35] and Sonar [17]. For ν, I wanted to look at only two values: 0.1 and 0.05. These values are used in the `mboost` implementation [21] as defaults or in examples, hence I only wanted to look at them. Their justification is that ν should be "small" in order to avoid overshooting the minimal empirical risk, as using a small ν ensures that the estimated effects increase slowly enough. This rationale was originally explained by Bühlmann and Hothorn in 2007 [7], who write that "a smaller

value of ν typically requires a larger number of boosting iterations and thus more computing time, while the predictive accuracy has been empirically found to be potentially better and almost never worse when choosing ν "sufficiently small" (e.g., $\nu = 0.1$)". They also came up with saying that a line-search for estimating ν in every iteration, as Friedman [16] proposed, is unnecessary for achieving a good performance and therefore suggested to use a constant value for ν.

For ϵ, I wanted to have a larger set of alternatives, since this is a parameter I introduced with this new method. Therefore it made sense to spend some more time on exploring it. I had already done some exploring in advance to find a reasonable range of values for ϵ. For that reason I decided to have a look at four values: 0.05, 0.01, 0.005 and 0.001. In combination I applied eight different settings of the learner to seven different data sets, four regression and three classification data sets. The following table shows the results expressed by ranks, where "1" is the best and "8" is the worst for a particular data set.

Data Set	ν	0.1	0.1	0.1	0.1	0.05	0.05	0.05	0.05
	ϵ	0.05	0.01	0.005	0.001	0.05	0.01	0.005	0.001
Airquality		7	4	3	1	8	6	5	2
Boston Housing		7	5	3	1	8	4	6	2
Wine		7	3	1	4	8	6	5	2
Kin8nm		7	2	3	5	8	6	1	4
Spam		7	5	3	1	8	6	4	2
Pima		4	1	3	7	8	5	2	6
Sonar		5.5	5.5	7	1	8	2	4	3
Mean rank		6.4	3.6	3.3	2.9	8	5	3.9	3
Median rank		7	4	3	1	8	6	4	2

Table 1: Benchmark for ifferent combinations of hyperparameters ν and ϵ, compared by mean rank and median rank.

In the end I calculated the mean rank of each setting. As result, I saw that the combination $\nu = 0.1$ and $\epsilon = 0.001$ achieved the lowest mean rank and lowest median rank, which is why this combination will be used as default for the two parameters.
However what should be recorded here is, that setting ϵ to 0.001 often results in a distinctly longer run-time compared to larger ϵ values, which is why in a case where users want fast results, they should consider increasing ϵ to 0.005.

4.5.2 Second Stage Model

For the second stage, one is given the choice between tree stumps and penalized splines (P-splines). As I later perform a benchmark study, where I compare

two versions of our method, one using tree stumps in the second stage and one using P-splines, to other methods, it seemed reasonable to set a default for the model in the second stage subsequently. However, as I want to decide about this default value at this point, I use a small part of the results already here. For 12 data sets, six regression and six classification tasks, I received the following results: the method using tree stumps performed better in nine cases, whereas the method using P-splines performed better in only three cases. Hence, I chose tree stumps as default method for the second stage of our method.

5 Visualization

5.1 Interpretability Measures

Christoph Molnar et al. [31] propose three model-agnostic measures of machine learning model interpretability, which reflect three dimensions of interpretability:

- the number of features used by the model
- the interaction strength
- the average complexity of the feature main effects

For the number of features used, the authors' definition is that a particular feature is regarded as "used" when a change in this feature results in a change in the prediction. For our method a feature will be counted as "used" when it gets selected during an iteration, which at the same time means that using this feature decreases the risk.

For the interaction strength I adapted the definition from [31] to our method. There, the interaction strength is defined as

$$IAS = \frac{\mathbb{E}(L(f, f_{ALE1st}))}{\mathbb{E}(L(f, f_0))}$$

where f is the final prediction function, f_{ALE1st} is the (ALE) main effect model and f_0 is the main of the predictions.
I transferred this definition to our model in the following way: f will be the final model after the fourth stage. Our main effect model (f_{ALE1st}) will be the model after the second stage, as up to that point no interactions are included. And f_0 will be the intercept model. The fractions in the formula are then calculated as the differences in the absolute risk values after the corresponding stages:

$$IAS = \frac{Risk_{Stage2} - Risk_{Stage4}}{Risk_{InterceptModel} - Risk_{Stage4}} \geq 0$$

Knowing the interaction strength of a model, brings along the ability of assessing

the expressiveness of individual feature effects, as generally the less interactions one has, the more reliable are the feature effects, e.g. when displayed in partial dependence plots.

Lastly, the authors propose a main effect complexity measure MEC for the ALE main effects, defined as the number of parameters that are necessary to approximate the feature curve with linear segments. The approximation has to reach an R^2 of at least 1 - an error ϵ. Their approach is to greedily set slopes of the linear segments to zero until the R^2 drops below the predetermined value. I applied the MEC measure to first two stages of our method.

The interpretability measures proposed above will be part of the visualization methods that will be described in the following sections.

5.2 Data Example

Throughout the visualization part, I want to use a data example in order to illustrate all functionalities of our method. For this purpose I chose the *Airquality* data set [10] that contains the daily air quality measurements in New York from May to September 1973. The goal is to predict the ozone in the air using information such as weather and time. First, I want to give a short overview over the features in the data set:

- **Ozone**: the mean ozone in parts per billion (ppb)
- **Solar.R**: the solar radiation
- **Wind**: the average wind speed in miles per hour
- **Temp**: the maximum daily temperature in degrees Fahrenheit
- **Month**: the month (1-12)
- **Day**: the day of the month (1-31)

5.3 Global Interpretability

I will start the visualization section with interpretability on a global level.
Through the section I will use the terms of training objects and test objects, where training objects are the objects that are returned by the main method after fitting, and test objects are the ones returned by the prediction method, basically the predicted values.

On a global level, our visualization method includes the following functionalities: it is able to display the feature main effects graphically and by means of a complexity measure, combined in the `featureplot.icb` function. Furthermore it comes with an overview of the impact of the individual stages and, in doing so, is able to express the interaction strength. Lastly, throughout the

boosting iterations, the course of the risk can be displayed along with the number of features used in the model. These functionalities are combined in the `riskplot.icb` function.
Here is how the two functions are defined:

```
riskplot.icb <- function(icb_object = NULL,
                         pred_object = NULL,
                         multiple = FALSE,
                         data_subset = NULL,
                         type = c("table","barplot",
                         "histogram"),
                         plot.which = c("Loss","Prediction"),
                         data_names = NULL,
                         fcount = FALSE)


featureplot.icb <- function(icb_object,
                            type = c("pdp","ale"),
                            col = "black",
                            feature = NULL,
                            data = NULL)
```

5.3.1 Partial Dependence Plots

The `featureplot.icb` function is able to display partial dependence plots for a trained `icb_object` based on the first two stages using `type` = "`pdp`". The method goes through every feature in the data and checks for the first two stages if the feature was included in the model. If that is the case, the method uses the `mboost` plot function to create corresponding partial dependence plots. If a feature is included in the first and the second stage, the estimated effect from the first as well as from the second stage are displayed in a plot. Since the main method works with an offset, the linear effect is added on top of the non-linear effect in order to have both on a consistent scale.

The functionality of the `mboost` plot method is the following: for numeric features, it plots the sorted feature values on the x-axis and the corresponding predictions on the y-axis, whereas for categorical features, simply one prediction value per category is plotted.

Now let's have a look at the data example. Figure 4 shows the effect of the solar radiation (`Solar.R`) on the predicted ozone. The linear curve shows the estimated effect after the first linear stage of the model. The black step function shows the effect estimated using tree stumps in the second stage, the red curve shows the feature effect using P-splines.
One overall sees that increasing the solar radiation increases the effect of the feature on the predicted ozone value. For solar radiation values below 200 the

effect on the outcome is negative, for values above it is positive. The smooth red curve shown for the model using P-splines is slightly S-shaped, whereas the effect for the model using trees shows two clear jumps.

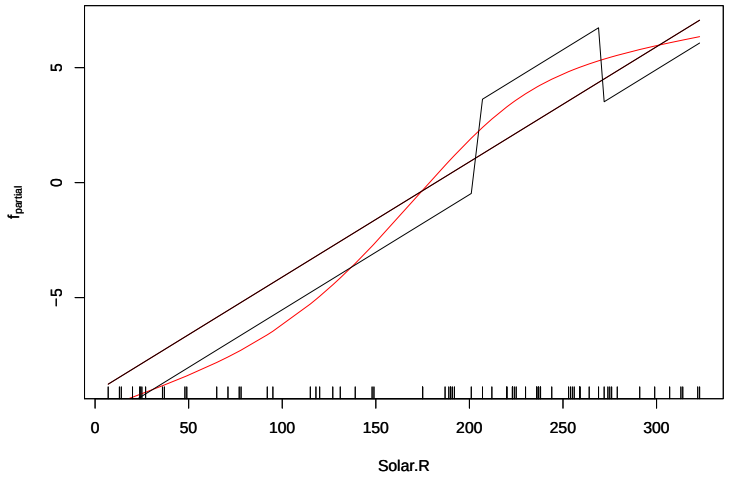

Figure 4: Partial Dependence Plot of Solar.R for the Airquality data set using the Interpretable Component-wise Boosting Method. The black step function shows the estimated non-linear effect for the model using tree stumps, the red curve for using P-splines.

5.3.2 ALE Main Effect Complexity

Using the definition from [31], I apply the ALE main effect complexity measure to the first two stages of our method. For a particular feature it is first checked whether the feature is included in the non-linear stage and if so, a corresponding ALE plot is created. If it is not included there, the linear stage is regarded, and depending on whether the feature is included here, an ALE plot is created.
The plot shows the estimated feature effect as well as the complexity `C`, measured in the number of parameters needed to estimate the model sufficiently. Accordingly, the R-squared `R2` is displayed as well the variance V of the particular ALE main effect.
Figure 5 visualizes the complexity of the effect of the solar radiation. The complexity `C` of 7 indicates that seven parameters are necessary to estimate the feature curve with step-wise linear functions and reach an R-squared of at least 0.99. The scale difference compared to 4 originates from the fact, that here only the second stage is included

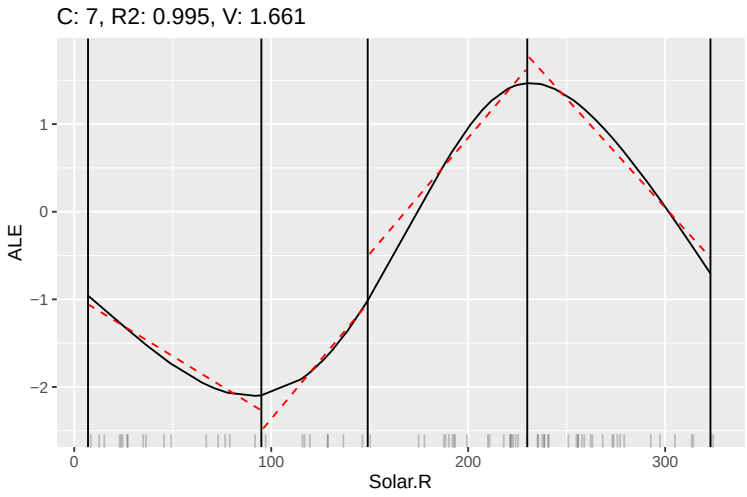

Figure 5: ALE Plot of Solar.R for the Airquality data set.

5.3.3 Risk Tables

A main visualisation tool introduced with this new method are the risk tables. The tables show the risk behaviour through the four stages. There are two different risk tables, which are explained here.

The first one serves as an overview table of the risk for one specific data set and can be created for both train and test objects. The table shows mean values for the absolute risk values (either in terms of mean squared error or mean misclassification error), the relative risk explanation per stage as well as the cumulative risk explanations. In addition it shows the number of iterations that were used for each stage, as well as the interaction strength as defined earlier. As baseline serves the risk that is left after the intercept model was fitted. The relative risk explanations then have to be seen in relation to the remaining risk after the intercept model was applied. So one can say for example, that "the first stage is able to explain *xx.xx* % of the left over risk from the intercept model".

Below, a risk table (6) for a training object is displayed for the Airquality data set. One sees that the linear stage is already able to explain 63.03% of the risk that is left after the use of an intercept model. The P-splines in the second stage are able to explain additional 13.16%. The trees of depth two thereupon can only reduce the risk a little bit (2.72%), whereas the fourth stage that uses deeper trees up to depth four is able to reduce the amount by an additional 13.77%. This leads to the assumption that interactions which are more complex than degree two exist in the data.

For comparison, I also want to look at the prediction table (7). Here, the overall relative risk explanation in the end is almost 20% lower than for training. Especially the deeper trees in the last stage are not able to explain a similar percentage of the risk, only 4.21% compared to 13.77% for training. Consequently, the same impression is received from the interaction strength measure which is 0.178 for the training, but only 0.089 for the test data. This is why I think the presumed complex interactions may not be existing in such a strong way and the training model may have slightly overfitted.

	Risk left	%-Risk Explanation	Cumulative %-Risk Explanation	Number of iterations
Intercept Model	1022.53			1
Phase 1: Linear	378.08	63.03 %	63.03 %	33
Phase 2: Splines/Tree Stumps	243.55	13.16 %	76.18 %	30
Phase 3: Trees (depth=2)	215.74	2.72 %	78.9 %	12
Phase 4: Deeper Trees	74.92	13.77 %	92.67 %	63
Interaction strength	0.178			

Figure 6: Training Risk Table for the Airquality Data Set.

	Risk left	%-Risk Explanation	Cumulative %-Risk Explanation	Number of iterations
Intercept Model	1256.13			1
Phase 1: Linear	562.29	55.24 %	55.24 %	33
Phase 2: Splines/Tree Stumps	423.4	11.06 %	66.29 %	30
Phase 3: Trees (depth=2)	394.68	2.29 %	68.58 %	12
Phase 4: Deeper Trees	341.78	4.21 %	72.79 %	63
Interaction strength	0.089			

Figure 7: Test Risk Table for the Airquality Data Set.

The second risk table is able to show the relative risk explanation of every stage for different data sets. Again, one is able to have a look into either training

objects or a test objects and can specify the names of the data sets that will be shown as the row names of the displayed table. The column for the overall relative risk explanation is color coded, where dark greens represent higher relative risk explanation and lighter greens represent lower relative risk explanation. Figure 8 shows a comparison table for the three data sets Airquality, Boston Housing and Bike.

Test Data %-Risk Explanation	Stage 1 (Linear)	Stage 2 (Non-linear)	Stage 3 (Trees of depth 2)	Stage 4 (Deeper Trees)	Overall %-Risk Explanation
Airquality	55.24 %	11.06 %	2.29 %	4.21 %	72.79 %
Boston Housing	68.21 %	10.39 %	6.64 %	1.56 %	86.8 %
Bike	41.79 %	0.59 %	14.43 %	10.71 %	67.51 %

Figure 8: Test Risk Table for the Airquality, Boston Housing and Bike Demand Data Sets.

5.3.4 Risk Plots

One additional feature I added to the main method is a counter for the number of features that are used by the model. In each iteration it is checked whether a feature is already included in the model or if it is used for the first time. If that is the case, the feature counter for that iteration is increased by one.
The feature counter can be displayed along with the risk curves, which show the development of the risk over the iterations for the training and the test data.
In plot 9 the risk curves for the training data (in red) and the test data (in blue) for the Airquality data are displayed as well as the counter for features. The gray vertical lines represent the iteration where the transitions from one stage to the next take place. The model uses overall four features which are already included at the end of the first stage. Strong drops of the risk can be seen especially at the transition from the linear to the non-linear stage. Around iteration 100 the test risk stops to decrease and even slightly increases again, whereas the training risk continues to decrease. Hence, this would be a good point to stop to avoid overfitting.

5.4 Local Interpretability

In terms of local interpretability, single observations can be visualized using loss tables as well as barplots.

5.4.1 Loss Tables

The loss table introduced here is able to show the loss explanation on an observation level. The user has the possibility to choose certain instances from the

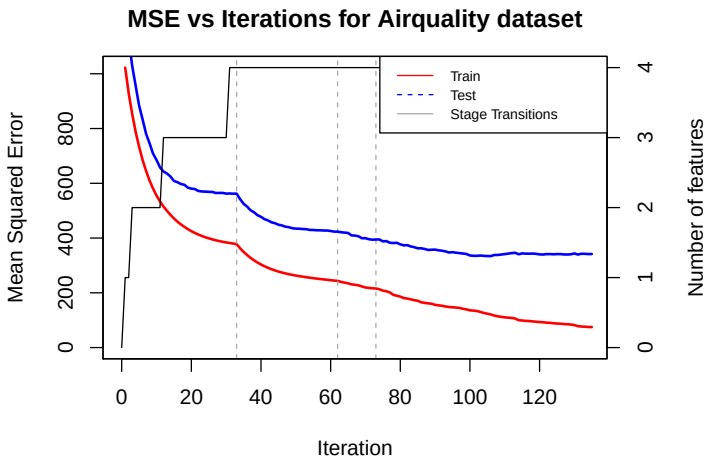

Figure 9: Risk Curves and Feature Counter for the Airquality Data Sets.

test data. For those instances, the function then displays the relative amount of the loss that is explained in each of the four stages. The percentage displayed always has to be seen in relation to the previous column respectively the intercept model for the first stage. The "Linear" column therefore shows the percentage of the loss of the intercept model that is explained after the linear stage. The "Non-linear" column then displays the amount of the loss explained by the second stage in comparison to the "Linear stage", and so on.
In table 10, the loss explanation for the observations 1, 7, 27 and 33 from the test set of the Airquality data are displayed. For the first observation, one can actually see negative values in the first column. This means that after the linear stage actually less of the loss is explained for this observations compared to the intercept model. However, one then sees that especially in the non-linear stage but also in the third stage a big part of that loss is explained. For observation 7 the linear stage is able to explain almost all of the loss, and adding more stages even reduces the explained amount of loss. For observation 27 the loss is decreased throughout every stage. For observation 33 the first stage again is able to explain a huge amount of the loss. However, the second stage of the model increases the loss again dramatically. Even though the third and the fourth stage are then able to reduce the loss, the intercept model had explained more of the loss of this observation than the final model after the fourth stage.

5.4.2 Barplots

In addition to the loss table, I created a function to display two different barplots for the individual interpretability of our method.
The first one shows the absolute risk improvement per stage for different obser-

Individual %-Loss Explanation	Stage 1 (Linear)	Stage 2 (Non-linear)	Stage 3 (Trees of depth 2)	Stage 4 (Deeper Trees)	Overall %-Loss Explanation
Obs. 1	-4.34 %	55.7 %	8.35 %	-1.18 %	58.53 %
Obs. 7	97.84 %	-3.5 %	-1.71 %	-1.42 %	91.21 %
Obs. 27	40.56 %	17.1 %	3.41 %	8.28 %	69.34 %
Obs. 33	86.92 %	-132.38 %	24.15 %	12.12 %	-9.19 %

Figure 10: Individual Loss Table for the Airquality Data Set.

vations.
In plot 11 one again has a look at the observations 1, 7, 27 and 33. For observation 1, one can see that the overall loss is already very small here after the intercept model. The biggest loss improvement then is done in the non-liner stage. For observation 7, the linear stage of the model achieves a huge absolute loss reduction. For the later stages the loss then increases again to a very small absolute amount. Observation 27 has the hugest absolute loss left after the intercept model compared to the other three observations displayed here. The linear as well as the non-linear stage are then able to decrease huge amounts of the loss. Also the third and fourth stage still explain a noticeabe amount of the loss. For observation 33 one can again see that the linear stage reduced the loss, but the non-linear stage increases it again to an even higher level than before, that also the third and the fourth stage are only able to reduce slightly.
The barplot is especially interesting in combination with the risk table that shows the relative loss improvement. For example, for observation 27 the model is only able to reduce the overall loss by 69.34%. However, compared to other observations this is a quite huge absolute loss reduction.

A second barplot visualizes how the individual predictions change over the different stages. A dotted line shows the intercept model prediction and four bars per observation show the corresponding prediction after the four stages.
For the Airquality data example, one can see in plot 12 that for the first and the seventh observation the predictions strongly decrease in the linear stage in comparison to the intercept model. The following stages then increase the prediction again slightly. For the observations 27 and 33 one can see a contrary picture. Here, the predictions strongly increase in the linear stage in comparison to the intercept model. For observation 27, the prediction keeps increasing with the further stages, whereas for observation 33 the prediction noticeably increases in the non-linear stage, but afterwards decreases in the third and fourth stage.

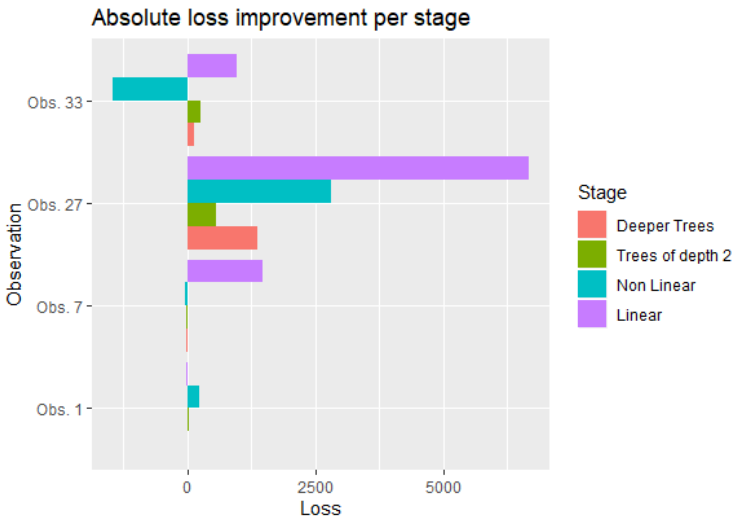

Figure 11: Absolute individual loss improvement per stage for the Airquality Data Set.

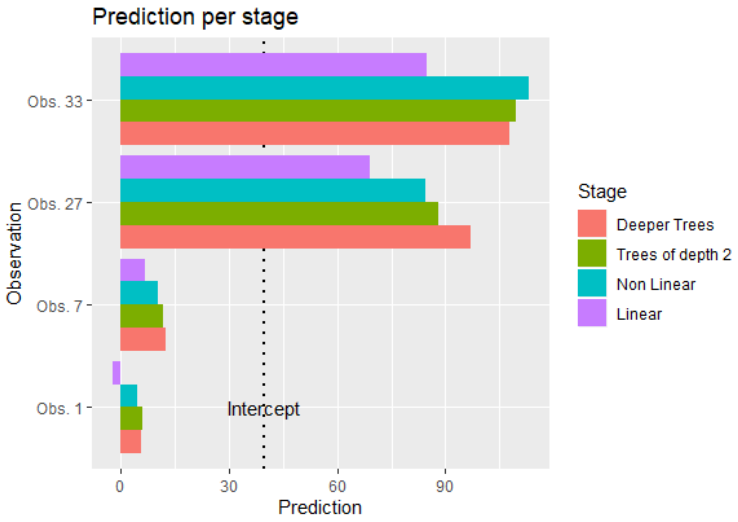

Figure 12: Prediction per stage for the Airquality Data Set.

6 Benchmark

Finally, I did a benchmark study in order to compare the performance of our new proposed method to different models.

6.1 Choice of Data Sets

The choice of an appropriate pool of data sets is crucial for the informative value of a benchmark. The data sets should not simply be selected randomly, but according to some deliberate pre-defined criteria. These criteria can be the following:

- The number of target classes in classification
- The availability of the data set on a certain platform
- Are the data sets especially interesting in the topic of interpretability?
- The number of features overall as well as the number of numeric / categorical features
- The number of observations

To reasonably test our method, I want to choose data sets that fulfill the following conditions based on the previous defined criteria.

For the regression benchmark, I selected the data sets based on the following criteria:
The data sets should be available on OpenML [9], an open machine learning environment with good connectivity to mlr. The range for the number of instances in a data set was chosen to be from 500 to 10000 and the number of features in a data set was determined to be between 5 and 30, both to limit computational effort. Also data sets with missing values were excluded.
From the remaining 67 data sets I chose three data sets with only numeric features as well as three data sets with numeric and categorical features, as I want to have a look at the performance of the method for both feature types. I try to use the full capacity of the pre-defined ranges regarding the number of instances and features.

Here is an overview over the regression data sets I chose in accordance with the criteria:

Name	Observations	Numeric. features	Categ. features
boston	506	12	2
cps_85_wages	534	4	7
credit-g	1000	7	14
kin8nm	8192	9	0
puma8NH	8192	9	0
wine_quality	6497	12	0

For classification, one only wants to have binary classification tasks, as especially mboost is only able to handle binary classification and our method is

built as a wrapper around it. Apart from that the same criteria were applied for the choice of the data sets as for the regression tasks. Again, I chose three data sets with only numeric features as well as three with numeric and categorical features.

Name	Observations	Numeric. features	Categ. features
bank-marketing	4521	7	9
car	1728	0	6
churn	5000	16	4
delta_elevators	9517	6	0
JapaneseVowels	9961	14	0
pollen	3848	5	0

6.2 Comparison with other Methods

I separated the benchmark for classification and regression tasks. For each of them, I want to have a look at six different data sets as described in an earlier section. For both types I compare our method to several different methods as listed below:

- Linear / Logistic Regression (from the `stats` package [34])
- GamBoost (from the `mboost` package [24])
- GLMBoost (Boosting for GLMs using the `mboost` package [24])
- Decision Trees (from the `rpart` package [38])
- Random Forests (from the `randomForest` package [26])
- eXtreme Gradient Boosting (from the `xgboost` package [12])
- Support Vector Machines (from the `kernlab` package [25])

In the further course of this thesis, I also refer to the latter three (Random Forests, eXtreme Gradient Boosting and Support Vector Machines) as state-of-the-art methods or predictive methods.

I use the `mlr` package [2] to run the benchmark. Furthermore, the benchmark is performed using Nested Resampling. Nested Resampling is a procedure to obtain unbiased performance estimates. In the case where one performs parameter tuning, one has two nested resampling loops. In the first, the outer resampling loop, the data is splitted in training and test sets. On these "outer" training sets the parameter tuning is done, using an inner resampling loop. After selecting the hyperparameters the learner is fitted on each outer training set

using the corresponding selected hyperparameters and its performance is evaluated on the "outer" test sets [3].
For the inner loop, where the tuning is performed, I used two-fold cross-validation, for the outer loop five-fold cross-validation. The hyperparameter tuning was done using Random Search with 30 iterations.

The tuning was performed for the following parameters. For our own method, I decided to use two versions of it, one that uses tree stumps (`icb.t`) in the second stage and one that uses P-splines (`icb.s`). Then I tuned the `nu`, `epsilon` , `max_depth` and for the spline version also the `df_spline` parameter.
For GAMBoost and GLMBoost, I tuned the learning rate `nu` as well as the `mstop` parameter, which decides over the number of iterations the models run.
For the decision trees I tune the `max_depth` and the `cp` parameter, which controls the tree complexity.
For the Random Forests, `mtry`, that determines the number of variables randomly sampled as split candidates, `nodesize`, which defines the minimum size of the terminal nodes, and `ntree`, that decides the number of trees to grow, are tuned.
For the xgboost, the maximum tree depth `max_depth` and the number of iterations `nrounds` are selected via tuning. And finally for the SVMs, the cost parameter `C` and the kernel width `sigma` are tuned.

So now let's have a look at the results for the regression tasks. The different learners are compared in terms of the Mean Squared Error (MSE). In table 2, the green background indicates the learner with the best performance for a particular data set, and the red background the one with the worst performance. For reasons of comparing, the gray background additionally marks the fifth best learner as representation of the median.

Name	icb.t	icb.s	lm	gamb	glmb	tree	rf	xgb	svm
boston	14.9	15.8	23.2	14.62	23.6	27.9	10.6	12.2	13.2
wages	19.1	18.8	19.2	19.4	18.6	20.5	20.1	19.5	19.4
credit-g *	3.45	3.53	3.59	3.55	3.47	5.14	3.59	3.64	3.67
kin8nm **	0.41	0.39	0.41	0.39	0.41	0.51	0.20	0.16	0.07
puma8NH	14.36	16.07	19.91	17.91	19.92	17.19	10.22	11.11	10.75
wine	0.55	0.53	0.54	0.52	0.54	0.61	0.36	0.41	0.50

Table 2: The Mean Squared Errors for the Regression data sets
* the MSE values here have to be multiplied by 10^6
** the MSE values here have to multiplied by 10^{-1}

The classification tasks in table 3 are compared using the Mean Misclassification Error (MMCE).

Name	icb.t	icb.s	logreg	gamb	glmb	tree	rf	xgb	svm
bank	0.100	0.102	0.097	0.102	0.096	0.109	0.010	0.101	0.107
car	0.005	0.007	0.047	0.053	0.047	0.054	0.017	0.000	0.004
churn	0.043	0.044	0.146	0.102	0.096	0.074	0.042	0.049	0.079
delta	0.116	0.118	0.119	0.117	0.119	0.164	0.119	0.119	0.119
JapVowels	0.012	0.013	0.036	0.024	0.039	0.051	0.014	0.006	0.004
pollen	0.064	0.073	0.265	0.058	0.241	0.132	0.073	0.066	0.061

Table 3: The Mean Misclassification Errors for the Classification Tasks

Regarding the absolute MSE and MMCE values, the two versions of our method perform quite similar for the most part. However, as already mentioned earlier, the version which uses tree stumps in the second stage shows slight advantages compared to the version that uses P-splines. Looking at the performance over the entire benchmark, our method is generally located in the middle to upper area. For most of the tasks, the best result is achieved by one of the state-of-the-art predictive methods Random Forests, eXtreme Gradient Boosting and Support Vector Machines. However, for two of the tasks, credit-g and delta, our method is able to perform best. Even though it often is not able to reach the lowest errors, it usually performs decently.

In order to gain a better overview and be able to make clearer statements about the learners' performances, the mean rank and median rank are now displayed for the nine different models (see table 4).

Task	Measure	icb.t	icb.s	lm/logreg	gamb	glmb	tree	rf	xgb	svm
Regression	Mean Rank	4.7	4.3	6.4	4.8	5.6	8.5	3.2	3.8	3.8
	Median Rank	4.5	4.8	6.8	4.3	6.8	9	2	2.5	3
Classification	Mean Rank	2.7	4.3	7.1	5.3	6.2	8	3.7	3.6	4.3
	Median Rank	3	4	7.5	6.3	6.8	9	4.5	4	4
Overall	Mean Rank	3.7	4.3	6.8	5	5.9	8.3	3.4	3.7	4.1
	Median Rank	3	4.3	7	4.8	6.8	9	3.5	3.5	3

Table 4: Benchmark performance comparison using mean and median rank.

Here, the advantage of our method using tree stumps over the version using P-splines again gets clear. For the regression tasks, the numbers are even, since the method using tree stumps has a lower median rank but a higher mean rank compared to its P-spline counterpart. A different picture unfolds when looking at the classification tasks, since here the method using tree stumps has clearly lower median and mean ranks.

The rank comparison to the rest of the learners yields two slightly different pictures. For the regression tasks, the three state-of-the-art prediction methods show clearly superior results, with Random Forests achieving the overall lowest mean and median rank. The two versions of our method are ranked behind, being on a same level with GAMBoost.
For classification though, the version of our method using tree stumps in the second stage, outperforms all others methods in terms of both mean and median rank.
Overall, the Random Forests still reach the lowest mean rank, however the gap to our methods is not a large one. Besides, our method using tree stumps shows the lowest overall median rank, being on the same level as the Support Vector Machines.

A further noticeable aspect of the benchmark study is, that our method often then shows superior results for a particular data set compared to the state-of-the-art methods, when also models like GAMBoost or GLMBoost show perform really well. This leads to the assumption that for those data sets interactions may not play a big role compared to the main feature effects, which can be estimated well by our method as well as GAMBoost and GLMBoost.
In an attempt to test this assumption, I want to have a look at the interaction strength in the particular data sets using the IAS measure from section 5 on test data predicted by the version of our method, which uses tree stumps in the second stage. Table 5 contains the results. The data tasks where the predictive methods clearly perform better than our method are displayed on the left side of the table, whereas the tasks where our method achieves similar or better results are displayed on the right.

Data Tasks with superior performance of predictive methods	IAS	Data Tasks with similar or worse performance of predictive methods	IAS
boston	0.055	wages	-0.080
kin8nm	0.5	credit-g	0.038
puma8NH	0.435	bank	0.037
wine	0.261	churn	0.230
car	0.291	delta	0.078
JapVowels	0.243	pollen	0.323

Table 5: Interaction strength for the data sets in the benchmark, calculated using the icb-method with tree stumps in the second stage.

Except for the boston data set, all tasks on the left side reveal higher values for the interaction strength than the ones on the right. This supports my assumption, that for those data sets, where interactions play a larger role, the predictive methods are able to outperform our method.

A possible explanation could be that the predictive methods, unlike our method, are able to estimate interactions right from the beginning. When giving our method the possibility to include interactions in the later stages, it does

not seem to be the case, that all interactions can be detected. A solution in cases with higher interaction strengths may be to lower the values for `epsilon` in order to keep the algorithm running for more iterations.

The better performance of our method for data tasks with less interactions on the other side may be explained as follows: as our method is already able to widely describe the existing linear and non-linear relationships in the first two stages and hereby reduces the risk to such an amount, the third and the fourth stage are only able to achieve small improvements. State-of-the-art methods, which include interactions from the beginning, might have a larger tendency of overfitting in these cases, which results in lower performance on test data.

7 Discussion

This thesis was aimed at creating a method which respects the trade-off between accuracy and interpretability in a reasonable way. Based on a stage-wise component-wise boosting procedure and using the `mboost` package, this method is able to achieve a performance that is generally better than for other interpretable models such as GAMBoost, Linear Regression and Decision Trees. In a benchmark study, for many data tasks the performance was equal or only slightly worse compared to state-of-the-art predictive models. The method may not always achieve the lowest errors, but performs very well in terms of ranks, which indicates consistent accuracy and the characteristic of rarely producing poor results.
Especially interesting are the benchmark results in comparison to GAMBoost as it also targets to solve the trade-off between accuracy and interpretability. Therefore it is nice to see that our method overall performs better, specifically for the classification tasks. Unfortunately, no direct comparison between our method and the GA^2M model and Double Penalty Model introduced in section 2 could be performed due to missing R implementations. However, an advantage of our method compared to GA^2M is the ability to not only include pair-wise interactions, but also interactions of larger orders.

An additional benefit of our method originates from its stage-wise manner. Since in the first two stages, only main effects and no interaction effects are estimated, the method seems to be less likely to include non-existing interactions and therefor less prone to overfitting. On the other hand, this results in comparatively lower accuracy in cases where many interactions occur, as other predictive methods include them right from the start.

Regarding interpretability, our method comes with a variation of additional tools. Besides visualizing estimated feature effects via Partial Dependence and Accumulated Local Effects Plots, the tools are able to show how the individual model parts contribute in terms of risk reduction as well as in terms of im-

pacting the prediction. Hence, the method enables drawing conclusions about which model complexity is necessary for a given data set, both in terms of main effect complexity and the degree of interactions. Additionally, the impact of the number of used features can be visualized by displaying it against the course of the risk throughout the iterations. In summary, our method respects all three dimensions of interpretability as proposed in [31].

Looking ahead, for some aspects further research and development can be considered. An important question for example was whether to strictly separate the stages or also allow models from earlier stages to be used in later stages. Here, I made the assumption, that all or the biggest part of the information that could be explained by a certain model type would be already covered in the corresponding stage, e.g. all linear effects in the first stage. By strictly separating the stages' model pools, one also avoids competition between simple and more complex models in one stage. Otherwise, it would be necessary to respect the degrees of freedom and keep them equal for all competing models, as if not, more complex models would have an advantage of being chosen compared to simpler models.
Implementing feature importance plots for the tree stages in our model is another aspect that can be thought about. As here, the method depends on the `blackboost` function, this was not feasible in the scope of this thesis.

Finally, I hope that the contents in this thesis are able to contribute to an increased consideration and usage of interpretable models in data science.

References

[1] Daniel W. Apley and Jingyu Zhu. *Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models*. 2016. arXiv: 1612.08468 [stat.ME].

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

[3] Bernd Bischl et al. *Nested Resampling*. URL: https://mlr.mlr-org.com/articles/tutorial/nested_resampling.html.

[4] L. Breiman et al. *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[5] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[6] Leo Breiman et al. *Classification and Regression Trees*. Statistics/Probability Series. Belmont, California, U.S.A.: Wadsworth Publishing Company, 1984.

[7] Peter Bühlmann and Torsten Hothorn. "Boosting Algorithms: Regularization, Prediction and Model Fitting". In: *Statist. Sci.* 22.4 (Nov. 2007), pp. 477–505. DOI: 10.1214/07-STS242. URL: https://doi.org/10.1214/07-STS242.

[8] Peter Bühlmann and B. Yu. "Boosting With the L2 Loss: Regression and Classification". In: *Journal of the American Statistical Association* 98 (Feb. 2003), pp. 324–339.

[9] Giuseppe Casalicchio et al. "OpenML: An R package to connect to the machine learning platform OpenML". In: *Computational Statistics* 32.3 (2017), pp. 1–15. DOI: 10.1007/s00180-017-0742-2. URL: http://doi.acm.org/10.1007/s00180-017-0742-2.

[10] John M. Chambers et al. *Graphical Methods for Data Analysis*. Wadsworth, 1983.

[11] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *CoRR* abs/1603.02754 (2016). arXiv: 1603.02754. URL: http://arxiv.org/abs/1603.02754.

[12] Tianqi Chen et al. *xgboost: Extreme Gradient Boosting*. R package version 0.90.0.2. 2019. URL: https://CRAN.R-project.org/package=xgboost.

[13] P. I. Corke. "A robotics toolbox for MATLAB". In: *IEEE Robotics Automation Magazine* 3.1 (Mar. 1996), pp. 24–32. ISSN: 1558-223X. DOI: 10.1109/100.486658.

[14] Paulo Cortez et al. "Modeling wine preferences by data mining from physicochemical properties." In: *Decision Support Systems* 47.4 (2009), pp. 547–553. URL: http://dblp.uni-trier.de/db/journals/dss/dss47.html#CortezCAMR09.

[15] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[16] Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine." In: *Ann. Statist.* 29.5 (Oct. 2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451. URL: https://doi.org/10.1214/aos/1013203451.

[17] T. J. Gorman R. P.; Sejnowski. "Learned Classification of Sonar Targets Using a Massively Parallel Network". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* (Jan. 1988). URL: http://papers.cnl.salk.edu/PDFs/Learned%5C%20Classification%5C%20of%5C%20Sonar%5C%20Targets%5C%20Using%5C%20a%5C%20Massively%5C%20Parallel%5C%20Network%5C%201988-3231.pdf.

[18] David Harrison and Daniel L. Rubinfeld. "Hedonic housing prices and the demand for clean air". In: *Journal of Environmental Economics and Management* 5.1 (1978), pp. 81–102. URL: https://EconPapers.repec.org/RePEc:eee:jeeman:v:5:y:1978:i:1:p:81-102.

[19] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, New York, 2009.

[20] Trevor Hastie and Robert Tibshirani. "Generalized Additive Models". In: *Statist. Sci.* 1.3 (Aug. 1986), pp. 297–310. DOI: 10.1214/ss/1177013604. URL: https://doi.org/10.1214/ss/1177013604.

[21] Benjamin Hofner et al. "Model-based Boosting in R: A Hands-on Tutorial Using the R Package Mboost". In: *Comput. Stat.* 29.1-2 (Feb. 2014), pp. 3–35. ISSN: 0943-4062. DOI: 10.1007/s00180-012-0382-5. URL: http://dx.doi.org/10.1007/s00180-012-0382-5.

[22] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. "Unbiased Recursive Partitioning: A Conditional Inference Framework". In: *Journal of Computational and Graphical Statistics* 15.3 (2006), pp. 651–674. DOI: 10.1198/106186006X133933. eprint: https://doi.org/10.1198/106186006X133933. URL: https://doi.org/10.1198/106186006X133933.

[23] Torsten Hothorn and Achim Zeileis. "Partykit: A Modular Toolkit for Recursive Partytioning in R". In: *J. Mach. Learn. Res.* 16.1 (Jan. 2015), pp. 3905–3909. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2789272.2912120.

[24] Torsten Hothorn et al. *mboost: Model-Based Boosting*. R package version 2.9-1. 2018. URL: https://CRAN.R-project.org/package=mboost.

[25] Alexandros Karatzoglou et al. "kernlab – An S4 Package for Kernel Methods in R". In: *Journal of Statistical Software* 11.9 (2004), pp. 1–20. URL: http://www.jstatsoft.org/v11/i09/.

[26] Andy Liaw and Matthew Wiener. "Classification and Regression by randomForest". In: *R News* 2.3 (2002), pp. 18–22. URL: https://CRAN.R-project.org/doc/Rnews/.

[27] Yin Lou et al. "Accurate intelligible models with pairwise interactions." In: *KDD*. Ed. by Inderjit S. Dhillon et al. ACM, 2013, pp. 623–631. ISBN: 978-1-4503-2174-7. URL: http://dblp.uni-trier.de/db/conf/kdd/kdd2013.html#LouCGH13.

[28] Tim Miller. "Explanation in Artificial Intelligence: Insights from the Social Sciences". In: *CoRR* abs/1706.07269 (2017). arXiv: 1706.07269. URL: http://arxiv.org/abs/1706.07269.

[29] Christoph Molnar. *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/. https://christophm.github.io/interpretable-ml-book/, 2018.

[30] Christoph Molnar, Bernd Bischl, and Giuseppe Casalicchio. "iml: An R package for Interpretable Machine Learning". In: *JOSS* 3.26 (2018), p. 786. DOI: 10.21105/joss.00786. URL: http://joss.theoj.org/papers/10.21105/joss.00786.

[31] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. *Quantifying Interpretability of Arbitrary Machine Learning Models Through Functional Decomposition*. Apr. 2019.

[32] Harsha Nori et al. "InterpretML: A Unified Framework for Machine Learning Interpretability". In: (Sept. 2019).

[33] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN: 1-55860-238-0.

[34] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2019. URL: https://www.R-project.org/.

[35] Ryan A. Rossi and Nesreen K. Ahmed. "The Network Data Repository with Interactive Graph Analytics and Visualization". In: *AAAI*. 2015. URL: http://networkrepository.com.

[36] Matthias Schmid and Torsten Hothorn. *Boosting Additive Models using Component-wise P-Splines*. 2008. URL: http://nbn-resolving.de/urn/resolver.pl?urn=nbn:de:bvb:19-epub-2057-7.

[37] Carolin Strobl, James Malley, and Gerhard Tutz. *An Introduction to Recursive Partitioning: Rationale, Application and Characteristics of Classification and Regression Trees, Bagging and Random Forests*. 2009. URL: http://nbn-resolving.de/urn/resolver.pl?urn=nbn:de:bvb:19-epub-10589-8.

[38] Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-15. 2019. URL: https://CRAN.R-project.org/package=rpart.

[39] G. Tutz and H. Binder. "Generalized additive modelling with implicit variable selection by likelihood based boosting". In: *Biometrics* to appear (2006).

[40] Wenjia Wang and Yi-Hui Zhou. *A Double Penalty Model for Interpretability*. 2019. arXiv: `1909.06263 [stat.ME]`.

Appendix

Code

The code developed for this master's thesis can be found on `https://github.com/LorenzHaller/InterpretableCompBoosting`.
Here, `Example.R` gives a broad overview over the method's functionalities. The train method is in `icb.R`, the predict method in `icb_predict.R` and the plot functions are in `icb_plot.R`. The scripts `icb_onehot.R` and `icb_onehot_predict.R` contain a different version of the method using one-hot encoding for categorical features.
Furthermore, `mlr_wrapper.R` contains a wrapper for the method that is able to use `mlr` functionalities. The corresponding benchmark setups are contained in `mlr_benchmark_regr.R` and `mlr_benchmark_classif.R`.

Declaration of Authorship

I hereby certify that the master thesis, titled "Gradually interpretable models via component-wise boosting", is entirely my original work except where otherwise indicated. I am aware of the University's regulations concerning plagiarism, including those regulations concerning disciplinary actions that may result from plagiarism. Any use of the works of any other author, in any form, is acknowledged adequately at their point of use.

Signature:

Date: