Modern approaches for component-wise boosting: Automation, efficiency, and distributed computing with application to the medical domain

Daniel Schalk

München 2023

Modern approaches for component-wise boosting: Automation, efficiency, and distributed computing with application to the medical domain

Daniel Schalk

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

eingereicht von
Daniel Schalk
am 07.12.2022

Erster Berichterstatter: Prof. Dr. Bernd Bischl
Zweiter Berichterstatter: PD Dr. Fabian Scheipl
Dritter Berichterstatter: Prof. Dr. Matthias Schmid

Tag der Disputation: 24.03.2023

Acknowledgments

This thesis would not have been possible without the help, support, guidance, and advice of many people! In particular, I would like to express my sincere gratitude to . . .

. . . *my supervisor Prof. Dr. Bernd Bischl for the seamless collaboration, trust, support, encouragement and many advice throughout the years.*

. . . *PD Dr. Fabian Scheipl and Prof. Dr. Matthias Schmid for their willingness to act as the second and third reviewer for my Ph.D. thesis.*

. . . *Prof. Dr. Christian Heumann and Prof. Dr. Helmut Küchenhoff for their availability to be part of the examination panel at my Ph.D. defense.*

. . . *Prof. Dr. Ulrich Mansmann for the pleasant cooperation and support via DIFUTURE.*

. . . *all my coauthors, especially David Rügamer, for the supportive collaboration.*

. . . *collegues who have become friends over time and with whom I had an unforgettable time (alphabetically ordered): Ben, Christoph, Flo, Giuseppe, Janek, Jann, Julia, Julia, Martin, Matthias, Moritz, Phillipp, Quay, Stefan.*

. . . *all current and former members of my working group: Thanks for always feeling welcome, awesome activities such as hiking, crafting, or playing cards, and, of course, drinking a lot of coffee.*

. . . *all remaining former and current colleagues at the Department of Statistics for the friendly atmosphere, thrilling soccer matches, nice retreats, and all other activities.*

. . . *my family and girlfriend who always support me, tolerate my quirks, and are always there for me.*

Summary

Component-wise boosting (CWB) with statistical models as base learners is a method at the intersection of statistics and machine learning. The advantages of CWB are its usability in high-dimensional feature spaces, the automatic (unbiased) variable selection, and the interpretability of the partial feature effects. However, a disadvantage of the method is the inefficient model structure, which often does not allow for processing vast amounts of data at desirable rates or requires too much memory. Furthermore, even though CWB is a user-friendly method, setting up the algorithm may require a high level of expertise.

The first part of this thesis addresses adapting CWB to the demands arising from these disadvantages. Thus, within this thesis, the internal structure of CWB was adjusted to increase its efficiency. First, the optimizer of CWB, a functional gradient descent, was extended to include Nesterov's momentum and accelerates CWB's fitting. Second, a more efficient data structure for numerical features reduces the required memory and further accelerates the fitting process. Furthermore, combining Automated Machine Learning (AutoML) concepts with CWB in a framework called Autocompboost helps users to benefit from CWB. Compared to other AutoML systems, Autocompboost focuses on the interpretability of the estimated model. The framework allows the user to assess the required model complexity and explain the decision-making process.

The second part of this work deals with the model estimation of CWB and the evaluation of binary classification models on distributed data. Here, the data are distributed across different sites and contain confidential information, making pooling of the data into one data set impossible. To fit CWB to distributed data, the fitting completely relies on aggregated data, so re-identifying individual data is impossible. Additionally, a server-specific adjustment of the estimated partial feature effects compensates for inhomogeneities in the individual data sets. Furthermore, updating evaluation procedures to this multi-site setup is essential. Thus, this thesis also deals with a technique to evaluate binary classification models by analyzing the receiver operating characteristics (ROC) curve on distributed data. The underlying method, computed in a distributed manner, is the ROC-GLM and allows an estimation of the ROC curve and the area under the curve with respective confidence intervals.

Finally, the implementations of the developed methods in the statistical programming language R conclude this dissertation. The package `compboost` contains an efficient CWB implementation, while the package `dsCWB` allows fitting CWB to distributed data. `Autocompboost` provides the AutoML framework for CWB and builds on `compboost`. Further, the package `dsBinVal` contains functionality for distributed ROC analysis.

Zusammenfassung

Komponentenweises Boosting (engl. component-wise boosting, CWB) mit statistischen Modellen als Basiskomponenten ist eine Methode, welche Eigenschaften aus der statistischen Modellierung und des Maschinellen Lernens in sich vereint. Vorteile von CWB sind neben der Praxistauglichkeit für hochdimensionale Daten die automatische (unverzerrte) Variablenselektion sowie die Intepretierbarkeit der Effekte der Einflussgrößen. Ein Nachteil der Methode ist die ineffiziente Modellstruktur. Diese erlaubt es nicht sehr große Datenmengen in adäquater Zeit zu verarbeiten. Zudem benötigt es Fachwissen für die richtige Einstellung des Algorithmus, obwohl CWB als anwenderfreundliche Methode gilt.

Ein Schwerpunkt dieser Arbeit ist es, CWB den Anforderungen, welche sich aus diesen Nachteilen ergeben, mit Hilfe moderner Techniken des Maschinellen Lernens entsprechend anzupassen. Eine Effizienzsteigerung des Algorithmus wird durch die Anpassung der internen Struktur an zwei Stellen erzielt und beschrieben. Der zugrundeliegende Optimierer, ein funktioneller Gradientenabstieg, wird für eine beschleunigte Modellschätzung um Nesterovs Momentum erweitert. Die Nutzung einer effizienteren Datenstruktur für metrische Einflussgrößen reduziert zusätzlich den benötigten Arbeitsspeicher und beschleunigt nochmals die Modellschätzung. Damit eine größere Anwenderschaft von CWB profitiert, werden Konzepte des Automatischen Maschinellen Lernens (AutoML) adaptiert und ein Framework namens Autocompboost vorgestellt. Der Fokus von Autocompboost liegt dabei auf der Automatisierung von Datenvorverarbeitung bis hin zum Schätzen von CWB sowie der Interpretierbarkeit des geschätzten Modells. Die dem Nutzer zur Verfügung gestellten Methoden erlauben es die nötige Modellkomplexität zu beurteilen und gewährleisten vollständige Transparenz beim Prognostizieren.

Der zweite Schwerpunkt dieser Arbeit behandelt die Modellschätzung von CWB sowie Evaluation von binären Klassifikationsmodellen auf verteilten Daten. In diesem Szenario sind die Daten auf verschiedenen Servern gespeichert und beinhalten vertrauliche Informationen. Eine zentrale Sammlung der Daten ist deshalb nicht möglich. Um dennoch ein Modell mit CWB zu schätzen, werden die Originaldaten so aggregiert und verarbeitet, dass eine Re-Identifikation von Individualdaten ausgeschlossen ist. Zudem erfolgt eine Server-spezifische Anpassung der geschätzten Effekte, welche zum Ziel hat Inhomogenitäten in den Daten pro Server auszugleichen. Des Weiteren wird eine Technik zur Evaluation von binären Klassifikationsmodellen mittels ROC Analyse auf verteilten Daten beschrieben. Die zugrunde liegende Methode, welche verteilt berechnet wird, ist das ROC-GLM und erlaubt neben der Schätzung der ROC Kurve auch eine Schätzung der area under the ROC curve und dazugehörigen Konfidenzintervallen.

Der praktische Zugang zu den beschriebenen Methoden ist über die statistische Programmiersprache `R` gegeben und ebenfalls Teil dieser Arbeit. Während `compboost` eine performante CWB Implementation beinhaltet kann mit `dsCWB` der Algorithmus auf verteilten Daten geschätzt werden. Die Software `Autocompboost` beinhaltet das AutoML Framework und baut auf `compboost` auf. Das Paket `dsBinVal` enthält die Funktionalität für die verteilte ROC Analyse.

Contents

Part I - Introduction and Background

CHAPTER 1

Overview

Over the past few years, the goal of machine learning (ML) has broadened from pure predictive performance to more specific tasks. Instead of modeling an output as precisely as possible, a different trend has arisen to interpret the model and understand its decision-making. Methods to explain complex models after they have been fitted are gathered under the umbrella term interpretable ML (IML; Molnar, 2020), also referred to as explainable artificial intelligence (XAI; Arrieta et al., 2020). Others argue that out-of-the-box interpretable models should be used if interpreting the final model is the goal (Rudin, 2019). A technique at the intersection of statistics and ML is component-wise (gradient) boosting (CWB; Bühlmann and Yu, 2003; Bühlmann et al., 2007), which produces interpretable models if statistical components are used. CWB internalizes the ML concept of rigorous risk minimization for model fitting and inherits the interpretability from the underlying statistical base components. Because of the interpretability capabilities, feasibility in high dimensional feature spaces, and automatic feature selection, CWB is often used in practical applications – such as oral cancer prediction (Saintigny et al., 2011), detection of synchronization in bioelectrical signals (Rügamer et al., 2018), or classifying pain syndromes (Liew et al., 2020a,b).

But, challenges that arise from the needs of our digital society must be addressed in order for CWB to remain a visible player in the world of ML. With the advent of big data, an algorithm nowadays should be able to process massive amounts of data efficiently. Additionally, helping non-experts apply CWB by automating the most critical parts helps users to understand and trust the fitted model and makes CWB accessible to a broader audience. Another challenge is model fitting and evaluation on distributed systems. In this setting, the data set is exclusively and non-disclosively shared over multiple sites. After providing the methodological foundations in Chapter 2, this thesis addresses the latter challenges and focuses on three major topics.

Efficiency – Increasing CWB's efficiency w.r.t. to runtime and memory consumption is the first major contribution. In addition to describing the effect of implementations on efficiency, Chapter 3 introduces motivations behind accelerating the fitting process and choosing more efficient data representations. Chapter 6 (contributing article Schalk et al. (2018)) contains details about a more efficient software for CWB, while Chapter 7 (contributing article Schalk et al. (2022a)) discusses two methodological improvements in detail to account for these needs.

Automation – Chapter 4 contains details about automated ML (AutoML). Therefore, a high-level view of essential parts of AutoML is given. Further, the accuracy–interpretability trade-off is discussed and used to argue for trustworthy AutoML. Chapter 8 (contributing article Coors et al. (2021)) introduces `Autocompboost`, an interpretable AutoML framework, alongside explanations of methods to help practitioners to understand the extracted information.

Distributed computing – Especially in the medical context, e.g., when data are collected collaboratively at several hospitals, preserving patient privacy is key. Hence, Chapter 5 outlines challenges and techniques of distributed computing regarding model fitting and evaluation. Chapter 9 (contributing article Schalk et al. (2023a)) then proposes a distributed version of CWB. Chapter 10 (contributing article Schalk et al. (2022b)) further describes methods to distributively analyze the receiver operating characteristic (ROC) curve. The methods are implemented in an `R` package `dsBinVal` (cf. Chapter 11, contributing article Schalk et al. (2023b)).

CHAPTER 2

Methodological background

The methodological details underpinning both the work presented in Chapters 3, 4, and 5 as well as the contributed articles are first introduced here. The basic ML terms and assumptions are introduced in Section 2.1, followed by details about CWB in Section 2.2. Elaborations on model evaluation in Section 2.3 and the distributed data setup in Section 2.4 complete this chapter.

2.1 Supervised machine learning

In supervised ML, a p-dimensional covariate or feature vector $\mathbf{x} = (x_1, \dots, x_p)$ originates from a feature space $\mathcal{X} = (\mathcal{X}_1 \times \dots \times \mathcal{X}_p) \subseteq \mathbb{R}^p$. Respectively, the response or outcome value y is sampled from a target space $\mathcal{Y}$. ML aims to find the unknown relationship $f : \mathcal{X} \to \mathbb{R}$. To that end, n observations $(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \dots, n$, are drawn (conditionally) independently from an unknown probability distribution $\mathbb{P}_{xy}$ on the joint space $\mathcal{X} \times \mathcal{Y}$ and assembled into a data set $\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\}$. An inducer, learner, or algorithm $\mathcal{I} : \mathbb{D} \times \Lambda \to \mathcal{F}$ with hyperparameters (HPs) $\boldsymbol{\lambda} \in \Lambda$ estimates the unknown relationship $f \in \mathcal{F}$ by calculating $\hat{f} = \mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D})$. Internally, the inducer minimizes the empirical risk $\mathcal{R}_{\text{emp}}(\hat{f} \mid \mathcal{D}) = n^{-1} \sum_{(\mathbf{x},y)\in\mathcal{D}} L(y, \hat{f}(\mathbf{x}))$ given a loss function $L : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$ and data set $\mathcal{D}$. This process is called empirical risk minimization. In the following, $\mathbf{x}$ and y denote arbitrary members of $\mathcal{X}$ and $\mathcal{Y}$. Additionally, throughout this thesis, the HPs are assumed to be fixed or automatically selected by wrapping the inducer with HP optimization (HPO; see, e.g., Feurer and Hutter, 2019) (unless stated otherwise).

2.2 Component-wise boosting

CWB uses an iterative algorithm to estimate a model that corresponds to a generalized additive model (GAM; Hastie, 2017)

$$g(\mathbb{E}[Y|\mathbf{x}]) = f(\mathbf{x}) = f_0 + \sum_{k=1}^{K} b_k(\mathbf{x}) \tag{2.1}$$

with link function g, random variable Y for the response with outcome y, offset $f_0 \in \mathbb{R}$, and base learners $b_k, k \in \{1, \dots, K\}$, to model the additive terms. The choice of the base learners varies from simple univariate linear regression to more complex learners such as trees. In this thesis, the choice of base learners is restricted to statistical models that are parametrized by base learners $b_k(\mathbf{x}|\boldsymbol{\theta}_k)$. This section addresses the base learners, explains the basic CWB algorithm, and discusses important properties of a model estimated with CWB. Further, the model defined in Equation (2.1) also allows the inclusion of tensor product base learners to account for interactions. Adding more flexibility also enables estimating generalized additive mixed models (GAMMs; see, e.g., Wood, 2017) and is further clarified in Chapter 5.

2.2.1 Base learners

The k^{th} base learner $b_k : \mathcal{X} \to \mathbb{R}$ is used to model the contribution of one or multiple features to f and is parameterized with $\boldsymbol{\theta}_k \in \mathbb{R}^{d_k}$, $d_k \in \mathbb{N}$. Each base learner uses a generic basis representation $g_k : \mathcal{X} \to \mathbb{R}^{d_k}$, $\mathbf{x} \mapsto g_k(\mathbf{x}) = (g_{k,1}(\mathbf{x}), \dots, g_{k,d_k}(\mathbf{x}))^\mathsf{T}$ and is linear in the parameters, i.e., $b_k(\mathbf{x}|\boldsymbol{\theta}_k) = g_k(\mathbf{x})^\mathsf{T}\boldsymbol{\theta}_k$. Given a data set with n observations, a design matrix $\mathbf{Z}_k := (g_k(\mathbf{x}^{(1)}), \dots, g_k(\mathbf{x}^{(n)}))^\mathsf{T} \in \mathbb{R}^{n \times d_k}$ is attached to the base learner b_k and is used to fit this base learner. Note that base learners are typically not defined on the whole feature space, but rather on a subset $\text{dom}(b_k) \subseteq \mathcal{X}$. For example, a common choice for CWB is to define p base learners with b_k modeling the k^{th} feature $x_k \in \mathcal{X}_k$ to estimate univariate contributions.

Four typical base learners mainly used in this work are (penalized) linear base learners, spline base learners, categorical or random effects base learners, and row-wise tensor product base learners. These base learners are outlined in more detail in Chapter 9. Further, most base learners require setting HPs $\boldsymbol{\lambda}_k$. For example, using a P-spline (Eilers and Marx, 1996) requires choosing the degree of the basis functions, the number of knots, the order of differences for penalizing, and a penalty parameter α_k that controls the smoothness of the spline. Hence, base learners are defined by their design matrix $\mathbf{Z}_k$ and a potential penalty matrix $\boldsymbol{K}_k \in \mathbb{R}^{d_k \times d_k}$. For most base learners, the penalty matrix $\boldsymbol{K}_k$ already includes α_k without explicitly stating so. For example, P-splines uses $\boldsymbol{K}_k = \alpha_k \boldsymbol{D}_k^\mathsf{T}\boldsymbol{D}_k$ with $\boldsymbol{D}_k$ representing the penalized differences or $\boldsymbol{K}_k = \alpha_k \boldsymbol{I}_{d_k}$ with identity matrix $\boldsymbol{I}_{d_k} \in \mathbb{R}^{d_k \times d_k}$ for ridge regression. Estimating $\boldsymbol{\theta}_k$ with a given response vector $\mathbf{y} = (y^{(1)}, \dots, y^{(n)})^\mathsf{T} \in \mathbb{R}^n$ and the L_2-loss $(L_2(y, b_k(\mathbf{x}|\boldsymbol{\theta}_k)) = (y - b_k(\mathbf{x}|\boldsymbol{\theta}_k))^2)$ yields the least squares estimator $\hat{\boldsymbol{\theta}}_k = (\mathbf{Z}_k^\mathsf{T}\mathbf{Z}_k + \boldsymbol{K}_k)^{-1}\mathbf{Z}_k^\mathsf{T}\mathbf{y}$.

2.2.2 Fitting algorithm

A model fit by CWB is an ensemble consisting of $M \in \mathbb{N}$ updates added iteratively. The iterative algorithm is initialized by setting the offset f_0 (cf. Equation (2.1)) as loss-optimal constant model $\hat{f}^{[0]} = \arg\min_{c \in \mathbb{R}} \mathcal{R}_{\text{emp}}(c|\mathcal{D})$. Further updates $\hat{b}^{[m]}$ in iteration $m \in \mathbb{N}$ are constructed according to gradient descent in function space with

$$\hat{f}^{[m+1]} = \hat{f}^{[m]} + \nu \hat{b}^{[m]} \tag{2.2}$$

and learning rate $\nu > 0$ to control the size of a model update. To calculate the model update $\hat{b}^{[m]}$, the functional gradient $\nabla_f L(y, f)$ is evaluated at the current model estimate $f = \hat{f}^{[m]}$ after m iterations. The negative functional gradient is expressed as pseudo residuals $r^{[m](i)} = -\nabla_f L(y^{(i)}, f(\mathbf{x}^{(i)}))|_{f=\hat{f}^{[m-1]}}$,

$i \in \{1, \dots, n\}$. Instead of adding $\boldsymbol{r}^{[m]} = (r^{[m](1)}, \dots, r^{[m](n)})^\mathsf{T}$ to $\hat{f}^{[m]}$ as a model update, the pseudo residuals are projected onto the column space spanned by the basis representation of the respective base learners $b_1, \dots, b_K$ by fitting them all to $\boldsymbol{r}^{[m]}$ w.r.t. the L_2-loss. Hence, the goal is to minimize the sum of squared errors (SSE), which results in the least squares estimator:

$$\begin{aligned}\hat{\boldsymbol{\theta}}_k^{[m]} &= \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^{d_k}} \sum_{i=1}^n \left(r^{[m](i)} - b_k(\mathbf{x}^{(i)}|\boldsymbol{\theta})\right)^2 \\ &= \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^{d_k}} \mathrm{SSE}_k(\boldsymbol{r}^{[m]}, \boldsymbol{\theta}) \\ &= (\mathbf{Z}_k^\mathsf{T}\mathbf{Z}_k + \boldsymbol{K}_k)^{-1}\mathbf{Z}_k^\mathsf{T}\boldsymbol{r}^{[m]} \end{aligned} \tag{2.3}$$

The base learner $\hat{b}^{[m]}(\mathbf{x}) = b_{k^{[m]}}(\mathbf{x}|\hat{\boldsymbol{\theta}}^{[m]})$ used for updating the model in iteration m with index $k^{[m]} = \arg\min_{k\in\{1,\dots,K\}} \mathrm{SSE}_k(\boldsymbol{r}^{[m]}, \hat{\boldsymbol{\theta}}_k^{[m]})$ is chosen as the base learner that minimizes the SSE. This updating scheme is repeated M times or until an early stopping criterion is met. This early stopping criterion can range from observing the risk improvement on a validation data set to simpler criteria, such as stopping after exhausting a time budget. Choosing a sufficiently small learning rate $\nu \in [0.01, 0.1]$ was shown to give fast convergence (see, e.g., Bühlmann et al., 2007). Algorithm 1 assembles all the steps required to fit CWB.

Algorithm 1 Vanilla CWB algorithm

Input Train data $\mathcal{D}$, learning rate ν, number of boosting iterations M, loss function L, base learners $b_1, \dots, b_K$
Output Model $\hat{f} = \hat{f}^{[M]}$ defined by fitted parameters or coefficient vectors $\hat{\boldsymbol{\theta}}^{[1]}, \dots, \hat{\boldsymbol{\theta}}^{[M]}$

```
1:  procedure CWB($\mathcal{D}, \nu, L, b_1, \dots, b_K$)
2:      Initialize: $f_0 = \hat{f}^{[0]}(\mathbf{x}) = \arg\min_{c\in\mathbb{R}} \mathcal{R}_{\text{emp}}(c|\mathcal{D})$
3:      while $m \leq M$ do
4:          $r^{[m](i)} = -\left.\frac{\partial L(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f(\mathbf{x}^{(i)})}\right|_{f=\hat{f}^{[m-1]}}, \quad \forall i \in \{1, \dots, n\}$
5:          for $k \in \{1, \dots, K\}$ do
6:              $\hat{\boldsymbol{\theta}}_k^{[m]} = \left(\mathbf{Z}_k^\mathsf{T}\mathbf{Z}_k + \boldsymbol{K}_k\right)^{-1}\mathbf{Z}_k^\mathsf{T}\boldsymbol{r}^{[m]}$
7:              $\mathrm{SSE}_k = \sum_{i=1}^n (r^{[m](i)} - b_k(\mathbf{x}^{(i)}|\hat{\boldsymbol{\theta}}_k^{[m]}))^2$
8:          $k^{[m]} = \arg\min_{k\in\{1,\dots,K\}} \mathrm{SSE}_k$
9:          $\hat{f}^{[m]}(\mathbf{x}) = \hat{f}^{[m-1]}(\mathbf{x}) + \nu b_{k^{[m]}}(\mathbf{x}|\hat{\boldsymbol{\theta}}_{k^{[m]}}^{[m]})$
10:     return $\hat{f} = \hat{f}^{[M]}$
```

2.2.3 Important properties

Restricting the base learners to be linear in the parameters enables adding up base learners b_k of the same type with different parameter vectors $\hat{\boldsymbol{\theta}}_k'$ and $\hat{\boldsymbol{\theta}}_k^*$ by $b_k(\mathbf{x}|\hat{\boldsymbol{\theta}}_k') + b_k(\mathbf{x}|\hat{\boldsymbol{\theta}}_k^*) = b_k(\mathbf{x}|\hat{\boldsymbol{\theta}}_k' + \hat{\boldsymbol{\theta}}_k^*)$. Hence, the estimated coefficient vector $\hat{\boldsymbol{\theta}}_k = \sum_{m=1}^M \mathbb{1}_{\{k\}}(k^{[m]})\hat{\boldsymbol{\theta}}_k^{[m]}$ is the sum of all estimated coefficients when b_k was selected. The selection in iteration m is expressed by the indicator function $\mathbb{1}_{\{k\}}(k^{[m]})$, which is

1 if $k^{[m]} = k$ and 0 otherwise. Computing $\hat{\boldsymbol{\theta}}_k$ gives an estimate of the partial feature effect by $b_k(\mathbf{x}|\hat{\boldsymbol{\theta}}_k)$ and allows interpretation. Note that interpreting the feature effect is possible for uni- and bivariate effects by interpreting the parameter estimates or visualizing the effect and becomes more challenging or impossible if the base learner models more than two features.

Further, the estimated coefficients $\hat{\boldsymbol{\theta}}_1, \dots, \hat{\boldsymbol{\theta}}_K$ converge to the maximum likelihood solution (see, e.g., Schmid and Hothorn, 2008) of Equation (2.1) for $M \to \infty$ if the loss L corresponds to a negative log-likelihood. Optimizing CWB conducts blockwise coordinate gradient descent updates by adjusting one coefficient vector $\hat{\boldsymbol{\theta}}_k$ as a block per iteration. Hence, CWB can be fitted in high dimensional feature spaces with $p \gg n$, since only one base learner must be fit to the pseudo residuals $\boldsymbol{r}^{[m]}$ at a time. For the same reason, an intrinsic feature selection is given by design because one base learner is either updated or newly added to the ensemble. The number of selected features depends on the number of iterations and requires early stopping to not select features based on fitting a base learner to noise. Additionally, incorporating regularization into all base learners can enforce equal degrees of freedom and yield a fair selection by restricting the flexibility of all base learners to be equal (cf. Chapter 7 (Appendix A.2) or Hofner et al., 2011).

2.3 Model evaluation

Throughout the contributing articles, evaluating models is essential for comparing different ML algorithms. Evaluating models w.r.t. their future performance on new unseen data consists of two parts: first, the performance measure, and second, the evaluation strategy.

2.3.1 Performance measure

The performance measure $\rho : \mathbb{R}^n \times \mathcal{Y}^n \to \mathbb{R}, \hat{\mathbf{y}}, \mathbf{y} \mapsto \rho(\hat{\mathbf{y}}, \mathbf{y})$ is a function to measure the similarity between predictions $\hat{\mathbf{y}} = (\hat{f}(\mathbf{x}^{(1)}), \dots, \hat{f}(\mathbf{x}^{(n)}))^\mathsf{T}$ and outcomes $\mathbf{y}$ of size n. For the sake of notation, throughout this section, $\tilde{\rho}$ is used as performance measure $\tilde{\rho}(\hat{f}_\mathcal{D}, \mathcal{D}^*) = \rho(\hat{\mathbf{y}}^*, \mathbf{y}^*)$ to better highlight the difference between a data set $\mathcal{D}$ used for training $\hat{f}_\mathcal{D} = \mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D})$ and a (potentially different) data set $\mathcal{D}^*$ to calculate ρ on $(y^*, \mathbf{x}^*) \in \mathcal{D}^*$ with $\hat{y}^{(i)^*} = \hat{f}_\mathcal{D}(\mathbf{x}^{(i)*}), i = 1, \dots, |\mathcal{D}^*|$. The performance measure ρ is task-specific and crucial for translating the practical endpoint into a quantitative measure. It is important to distinguish the performance measure $\tilde{\rho}$ from the empirical risk $\mathcal{R}_\text{emp}$ with loss function L. Point-wise loss functions, such as the loss L, can be used to construct a performance measure by setting $\tilde{\rho}(\hat{f}_\mathcal{D}, \mathcal{D}) = \mathcal{R}_\text{emp}(\hat{f}_\mathcal{D}|\mathcal{D})$, whereas not all performance measures imply a loss function. Since implementations are often restricted in their choice of L, it is more common to have a performance measure $\tilde{\rho}$ different from the empirical risk $\mathcal{R}_\text{emp}$. At the same time, evaluating a model is derivative-free and, hence, possible for arbitrary performance measures. It is noteworthy that having non-matching measures $\tilde{\rho} \neq \mathcal{R}_\text{emp}$ is one facet of the inductive bias (Gordon and Desjardins, 1995).

It is also important to differentiate between performance measures for specific tasks. For binary classification with $\hat{f}$ predicting a score, a threshold t is used to transform $\hat{f}$ into discrete classes $\hat{c}_t$ as a binary rule with $\hat{c}_t(\mathbf{x}) = \mathbb{1}_{]-\infty;t]}(\hat{f}(\mathbf{x}))$ (in the case of 0-1-encoding). Hence, a classifier $\hat{c}_t$ is based on the score $\hat{f}$ and the threshold t. The same holds for transforming scores $\hat{f}$ into $\hat{\pi} \in [0;1]$ to allow interpreting the predictions as probabilities[1]. Specific measures exist for each prediction type. For example, the accuracy, F1 score, or in general, all measures obtained from the confusion matrix (Stehman, 1997) rely on discrete classes $\hat{c}_t$. Evaluating $\hat{\pi}$ depends on the score $\hat{f}$ and a transformation to squash $\hat{f}$ into $[0;1]$. Measures based on $\hat{\pi}$, for example, are the Brier score (Brier et al., 1950) or the log loss. More comprehensive explanations can be found in Casalicchio (2019).

One measure based on scores $\hat{f}$ and subject of Chapter 10 is the area under the ROC curve (AUC). The AUC is a measure of discrimination and can be interpreted as the probability that the score of a randomly selected positive outcome has a higher score and hence a higher rank than the score of a randomly selected negative outcome (Fawcett, 2006). The ROC curve is built as a curve in the ROC space $(\text{FPR}(t), \text{TPR}(t))$ with threshold $t \in \mathbb{R}$, false positive rate $\text{FPR}(t) = \text{P}(\hat{f}(\boldsymbol{X}) > t \mid Y = 0)$, and true positive rate $\text{TPR}(t) = \text{P}(\hat{f}(\boldsymbol{X}) > t \mid Y = 1)$. In this probabilistic formulation, Y is the underlying random variable of a binary outcome y, and $\hat{f}(\boldsymbol{X})$ is also a random variable of an estimated score (Pepe, 2000) with random vector $\boldsymbol{X}$. An alternative formulation is to define the ROC curve as $\text{ROC}(t_0) = \text{TPR}(\text{FPR}^{-1}(t_0))$ with $t_0 \in [0;1]$. The AUC is then given as $\text{AUC} = \int_0^1 \text{ROC}(t_0)\, dt_0$. Estimating the ROC curve is done by calculating the empirical TPR and FPR as $\widehat{\text{TPR}}(t|\mathcal{D}) = n_1^{-1} \sum_{\mathbf{x} \in \mathcal{D} \mid y=1} \mathbb{1}_{\{1\}}(\hat{c}_t(\mathbf{x}))$ and $\widehat{\text{FPR}}(t|\mathcal{D}) = n_0^{-1} \sum_{\mathbf{x} \in \mathcal{D} \mid y=0} \mathbb{1}_{\{1\}}(\hat{c}_t(\mathbf{x}))$, with n_1 and n_0 as the number of ones and zeros in the binary response $\mathbf{y}$. Respectively, the AUC is estimated by using the empirical TPR and FPR in $\widehat{\text{AUC}} = \int_0^1 \widehat{\text{ROC}}(t_0)\, dt_0$ with $\widehat{\text{ROC}} = \widehat{\text{TPR}} \circ \widehat{\text{FPR}}^{-1}$.

Note that each task type requires specific performance measures suited for the task. For example, it is common practice for multiclass classification to have a one-vs-rest or one-vs-one approach to break the evaluation down to multiple binary problems, evaluate them, and return the average as performance measure (cf., e.g., Flach, 2012, Chapter 3.1).

2.3.2 Evaluation strategy

The model evaluation aims to estimate the generalization error (GE; cf., Bischl et al., 2012), i.e., the performance ρ of a model $\hat{f}$ trained on $\mathcal{D}$ when processing new unseen data. Calculating the estimate $\widehat{\text{GE}}_{\text{resub}} = \tilde{\rho}(\hat{f}_{\mathcal{D}}, \mathcal{D})$ with $\hat{f}_{\mathcal{D}}$ fit and evaluated on the training data $\mathcal{D}$ is often called resubstitution error. Because of fitting and estimating on the same data set, the resubstitution error does not represent the performance of the model predicting new data and is optimistically biased. The most straightforward technique to estimate the GE is to split the data randomly into $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ with $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{test}}$, $\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$, and $n_{\text{train}} = |\mathcal{D}_{\text{train}}|$, $n_{\text{test}} = |\mathcal{D}_{\text{test}}|$ observations in the respective data sets. This scheme is known as hold-out splitting with n_{test} as a fraction, e.g., 0.33, of n. The GE estimation is conducted by calculating $\widehat{\text{GE}}_{\text{holdout}} = \tilde{\rho}(\hat{f}_{\mathcal{D}_{\text{train}}}, \mathcal{D}_{\text{test}})$. Since this is a valid scheme, the disadvantage is that neither the evaluation nor the training uses all observations. Depending on the size of the training and test data set, holdout splitting can introduce a high variance or a pessimistic bias of the estimated

[1]After transforming $\hat{f}$ to $\hat{p}$, it is still a score that was calculated, often, without underlying probability theory to back up the interpretation as probabilities.

GE. Therefore, more complex schemes aim to balance this bias–variance trade-off by repeatedly splitting the data set $\mathcal{D}$ into multiple train and test data sets $\{(\mathcal{D}_{\text{train},k}, \mathcal{D}_{\text{test},k}) \mid k = 1, \dots, K\}$.

The performance measure is then evaluated for all train–test combinations, yielding K estimates $\widehat{\text{GE}}_k = \tilde{\rho}(\hat{f}_{\mathcal{D}_{\text{train},k}}, \mathcal{D}_{\text{test},k})$, $k = 1, \dots, K$. The overall estimate for the GE is typically aggregated by the average $\widehat{\text{GE}} = K^{-1} \sum_{k=1}^{K} \widehat{\text{GE}}_k$. Hence, the whole process of model evaluation requires defining the performance measure ρ and a resampling strategy. Prominent resampling strategies are sub-sampling, cross-validation, or bootstrapping. For further information about resampling strategies, nested resampling, and theoretical details, see Bischl et al. (2012).

2.4 Distributed computing

The last concept introduced for this thesis is distributed computing. Distributed computing is an ambiguous term and may mean executing computations in parallel, like heavy matrix operations (see, e.g., Choi et al., 1994; Buluc and Gilbert, 2008), or laying out computations to one or multiple servers (see, e.g., Bekkerman et al., 2011; Verbraeken et al., 2020). Throughout this work, the term “distributed computing” is used to mean computing on non-disclosed data sets that are shared over multiple sites and exclusively located at individual sites. Hence, the data are unavailable to the analyst or user.

Distributing data to multiple sites is usually distinguished between horizontally and vertically distributed data (cf, e.g., Li et al., 2016). This thesis exclusively addresses horizontally distributed data. For horizontally distributed data, the data set $\mathcal{D}$ is partitioned into S data sets $\mathcal{D}_s = \{(\mathbf{x}_s^{(1)}, y_s^{(1)}), \dots, (\mathbf{x}_s^{(n_s)}, y_s^{(n_s)})\}$, $s = 1, \dots, S$, with n_s observations. The j^{th} feature vector at site s is denoted by $\mathbf{x}_{s,j} = (x_{s,j}^{(1)}, \dots, x_{s,j}^{(n_s)})^\mathsf{T}$. Respectively, the response vector at site s is denoted by $\mathbf{y}_s$. Each observation $(\mathbf{x}_s, y_s) \in \mathcal{D}_s$ originates from a site-specific data distribution $\mathbb{P}_{xy,s}$ and is exclusively located at the corresponding site s. The data sets are mutually exclusive $\mathcal{D}_s \cap \mathcal{D}_l = \emptyset \; \forall l, s \in \{1, \dots, S\}, l \neq s$, and the union of all data sets give the whole data set $\mathcal{D} = \bigcup_{s=1}^{S} \mathcal{D}_s$. The distributed setup in this work requires a host (e.g., the analyst’s machine) to control the communication with and between all sites to conduct distributed computations. In the following, Section 2.4.1 discusses the term “privacy-preserving” in more detail. Two common approaches, data aggregation (Section 2.4.2) and differential privacy (DP; Section 2.4.3) are also concerned. Finally, the term lossless is defined in Section 2.4.4.

2.4.1 Privacy-preserving machine learning

Dealing with the host–site setup, site s is not allowed to communicate parts of its data set $\mathcal{D}_s$ to the host. Sending information from the sites to the host must ensure that reverse-engineering parts of the data at the host is not possible. Hence, the main challenge of distributed model fitting is to find an algorithm $\mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D}_1, \dots, \mathcal{D}_S)$ that can estimate $\hat{f}$ under certain restrictions. These restrictions are inferred by security mechanisms that must be applied to protect the data by guaranteeing the avoidance of privacy breaches. In this setup, the host is vulnerable because it collects all information, processes it, and sends it to the sites. Hence, if an unauthorized third party infiltrates the host, it must be ensured that no raw data are either directly or indirectly (e.g., via reconstruction) leaked. This algorithm is then called privacy-preserving. The methods used in this work to achieve privacy preservation are directly incorporated into the algorithm and allow it to use that algorithm also for other distributed data sets. It is noteworthy that methods like k-anonymity (Sweeney, 2002) or ℓ-diversity (Machanavajjhala et al., 2007) aim to anonymize data to use that anonymized data set for modeling. However, this always requires preprocessing data before using it with a specific algorithm.

2.4.2 Data aggregation

A possible means to allow communication of information is to share aggregated values. For example, one could define that communicating the sum of a feature is not problematic as long as it consists of at least q values. Then, each site calculates $a_s = \sum_{i=1}^{n_s} x_{s,j}^{(i)}$ and communicates a_s to the host if $n_s \geq q$. The host adds up all partial sums to obtain the overall sum $\sum_{s=1}^{S} a_s$. The value q is also known as a privacy level and is often used as a privacy parameter in frameworks such as DataSHIELD (Gaye et al., 2014). The privacy level describes the minimal number of values required for an aggregation to allow sharing of the result. Aggregating data is used in more complex routines, such as calculating a linear model (LM), to ensure privacy.

Example: Distributed LM – Consider a design matrix $\mathbf{Z}$ and response vector $\mathbf{y}$. The estimated coefficients in the LM are $\hat{\boldsymbol{\theta}} = (\mathbf{Z}^\mathsf{T}\mathbf{Z})^{-1}\mathbf{Z}^\mathsf{T}\mathbf{y}$. Karr et al. (2005) presented a secure distributed LM based on simple aggregations. They split $\mathbf{Z}$ and $\mathbf{y}$ into $\mathbf{Z} = (\mathbf{Z}_1^\mathsf{T}, \dots, \mathbf{Z}_S^\mathsf{T})^\mathsf{T}$ and $\mathbf{y} = (\mathbf{y}_1^\mathsf{T}, \dots, \mathbf{y}_S^\mathsf{T})^\mathsf{T}$ to decompose the cross-products $\mathbf{Z}^\mathsf{T}\mathbf{Z}$ and $\mathbf{Z}^\mathsf{T}\mathbf{y}$ into sums of site terms $\mathbf{Z}^\mathsf{T}\mathbf{Z} = \sum_{s=1}^{S} \mathbf{Z}_s^\mathsf{T}\mathbf{Z}_s$ and $\mathbf{Z}^\mathsf{T}\mathbf{y} = \sum_{s=1}^{S} \mathbf{Z}_s^\mathsf{T}\mathbf{y}_s$. The design matrix $\mathbf{Z}_s$ and response vector $\mathbf{y}_s$ are located at site s. For the distributed LM, they first calculate $\boldsymbol{F}_s = \mathbf{Z}_s^\mathsf{T}\mathbf{Z}_s$ and $\boldsymbol{u}_s = \mathbf{Z}_s^\mathsf{T}\mathbf{y}_s$ at site s. Next, both aggregations $\boldsymbol{F}_s$ and $\boldsymbol{u}_s$ are communicated to the host and summed up to $\boldsymbol{F} = \sum_{s=1}^{S} \boldsymbol{F}_s$ and $\boldsymbol{u} = \sum_{s=1}^{S} \boldsymbol{u}_s$. Finally, the parameter estimates are calculated by $\hat{\boldsymbol{\theta}} = \boldsymbol{F}^{-1}\boldsymbol{u}$, again at the host. In this procedure, the host only sees aggregated values and cannot reconstruct parts of the raw data if the privacy level is met. To account for aggregations outputting multiple values, e.g., here the parameter vector, the privacy level q must be adjusted by multiplying q with the number of parameters p. Algorithm 2 summarizes the distributed LM and extends it by adding a penalty matrix $\boldsymbol{K} \in \mathbb{R}^{p\times p}$.

2.4.3 Differential privacy

In general, DP allows sharing information about a data set without revealing information concerning individuals in that data set. In contrast to data aggregation, an approach by DP is to add noise to a value obtained by a randomized mechanism $\mathcal{M} : \mathcal{X} \mapsto \mathcal{Y}$ with domain $\mathcal{X}$ (e.g., $\mathcal{X} = \mathbb{R}^p$) and target domain $\mathcal{Y}$ (e.g., $\mathcal{Y} = \mathbb{R}$ in regression). DP aims to prevent $\mathcal{M}$ from attacks that aim to re-identify individuals who may be in the ground data set (Dwork et al., 2006b). DP was introduced by Dwork et al. (2006a) formally as (ε, δ)-DP, which is given if – for any subset of outputs $R \subseteq \mathcal{Y}$ – the inequality

$$P(\mathcal{M}(\mathbf{x}) \in R) \leq \exp(\varepsilon)P(\mathcal{M}(\mathbf{x}') \in R) + \delta \tag{2.4}$$

holds for two adjacent inputs $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. Multiple ways exist to define adjacent inputs. Throughout this thesis, adjacent inputs are identified by first converting two input vectors $\mathbf{x}$ and $\mathbf{x}'$ to a histogram

Algorithm 2 Distributed LM fitting with penalty. The line prefixes `[S]` and `[H]` indicate whether the operation is conducted at the sites (`[S]`) or at the host (`[H]`).

Input Site design matrices $\mathbf{Z}_1, \dots, \mathbf{Z}_S$, response vectors $\mathbf{y}_1, \dots, \mathbf{y}_S$ and an optional penalty matrix $\boldsymbol{K}$.

Output Estimated parameter vector $\hat{\boldsymbol{\theta}}_l$.

```
1: procedure distLM($\mathbf{Z}_1, \dots, \mathbf{Z}_S, \mathbf{y}_1, \dots, \mathbf{y}_S, \boldsymbol{K}$)
2:     for $s \in \{1, \dots, S\}$ do
3:         [S] $\mathbf{F}_s = \mathbf{Z}_s^\mathsf{T}\mathbf{Z}_s$
4:         [S] $\boldsymbol{u}_s = \mathbf{Z}_s^\mathsf{T}\mathbf{y}_s$
5:         [S] Communicate $\mathbf{F}_s$ and $\boldsymbol{u}_s$ to the host
6:     [H] $\mathbf{F} = \sum_{s=1}^S \mathbf{F}_s + \boldsymbol{K}$
7:     [H] $\boldsymbol{u} = \sum_{s=1}^S \boldsymbol{u}_s$
8:     [H] return $\hat{\boldsymbol{\theta}} = \mathbf{F}^{-1}\boldsymbol{u}$
```

representation $\tilde{\mathbf{x}} \in \mathbb{N}^p$ and $\tilde{\mathbf{x}}' \in \mathbb{N}^p$. Then, the inputs are considered to be adjacent if the ℓ_1-norm of $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{x}}'$ is equal to one: adjacent $\mathbf{x},\ \mathbf{x}' \ \Leftrightarrow\ \|\tilde{\mathbf{x}} - \tilde{\mathbf{x}}'\|_1 = 1$ (Dwork et al., 2014). In (ε, δ)-DP, the value of ε controls how much privacy is guaranteed. The value of δ is the probability that $(\varepsilon, 0)$-DP is broken (also known as ε-DP and the original definition proposed by Dwork et al. (2006b)).

Applying DP in ML, the objective is to protect the estimated model $\hat{f}$ by adding noise η. Hence, the randomized mechanism is given by $\mathcal{M}(\mathbf{x}) = \hat{f}(\mathbf{x}) + \eta$. In this work, the Gaussian mechanism (Dwork et al., 2014) is used to protect $\mathcal{M}$. Therefore, the noise η follows a normal distribution $\mathcal{N}(0, \tau^2)$. It has been previously shown (p. 471; Dwork et al., 2014) that the requirement for (ε, δ)-DP (Equation (2.4)) is satisfied if the variance is set to any value $\tau \geq c\Delta_2(\hat{f})/\varepsilon$ with $c^2 > 2\ln(1.25/\delta)$, $\varepsilon \in (0, 1)$, and $\Delta_2(\hat{f})$ the ℓ_2-sensitivity of $\hat{f}$ measured as $\Delta_2(\hat{f}) = \max_{\text{adjacent } \mathbf{x},\, \mathbf{x}'} \|\hat{f}(\mathbf{x}) - \hat{f}(\mathbf{x}')\|_2$. Thus, in practice, the ℓ_2-sensitivity of the model $\hat{f}$ is required to determine possible values for ε and δ. Then, the amount of noise η is controlled by choosing ε and δ and using $\tau = c\Delta_2(\hat{f})/\varepsilon$ with $c = \sqrt{2\ln(1.25/\delta)}$.

The introduction of noise is always accompanied by the drawback of an approximation error by using $\mathcal{M}$ instead of $\hat{f}$. Hence, using DP in practice (e.g., for a distributed ROC analysis, cf., Chapter 10 (contributed article Schalk et al. (2022b))) requires assessing if an approximating distributed algorithm is still usable or not (Jayaraman and Evans, 2019). This assessment relies on studying the effect of ε and δ on inaccuracies of a distributed algorithm.

2.4.4 Lossless distributed algorithms

As stated above for DP, estimating algorithms in a distributed and privacy-preserving fashion does not always allow finding an exact distributed pendant. Finding an exact algorithm in that context is also called lossless estimation (Luo et al., 2022). Throughout this work, a distributed fitting procedure fit on the distributed data sets $\mathcal{D}_1, \dots, \mathcal{D}_S$ is lossless if the model parameters of the original algorithm fitted on the pooled data set $\mathcal{D}$ are the same as for the distributed approach. For example, the distributed

LM distLM (Algorithm 2) gives the same parameter estimates $\hat{\boldsymbol{\theta}}$ as calculating the least squares estimator $\hat{\boldsymbol{\theta}}^* = (\mathbf{Z}^\mathsf{T}\mathbf{Z} + \boldsymbol{K})^{-1}\mathbf{Z}^\mathsf{T}\mathbf{y}$ on the pooled data set. Hence, $\hat{\boldsymbol{\theta}} = \hat{\boldsymbol{\theta}}^*$ induces that distLM is a lossless distributed algorithm of the LM. Respectively, if a function $\hat{f}(\mathbf{x}|\boldsymbol{\theta})$ is parameterized by $\boldsymbol{\theta}$, then the functions $\hat{f}(\mathbf{x}|\hat{\boldsymbol{\theta}})$ and $\hat{f}(\mathbf{x}|\hat{\boldsymbol{\theta}}^*)$ are equal, as are the empirical risks $\mathcal{R}_{\text{emp}}(\hat{f}(.|\hat{\boldsymbol{\theta}})|\mathcal{D})$ and $\mathcal{R}_{\text{emp}}(\hat{f}(.|\hat{\boldsymbol{\theta}}^*)|\mathcal{D})$.

CHAPTER 3

Efficiency

A downside of CWB that occurs in practice is its computational complexity w.r.t. runtime and memory consumption. For bigger data sets, this can be a reason not to use CWB. This chapter introduces the problem on two different levels. First, in Section 3.1, the state-of-the-art software of CWB, its drawbacks, and potential solutions on the software side are shortly introduced. Second, the effect of optimization routines and data representations are outlined in Section 3.2 and 3.3.

3.1 Implementations

The most commonly used software for CWB is `mboost` (Hothorn et al., 2010, 2020) and is implemented in the statistical software `R` (R Core Team, 2022). The implementation provides a wide range of base learners and is flexibly extendible. Thus, `mboost` was constantly extended by advanced techniques such as modeling functional data (Brockhaus et al., 2020), boosting location scale and shape models (Hofner et al., 2016), or probing (Thomas et al., 2017). However, one of the problems of `mboost` is the extensive information stored during fitting. Storing that information can not only exceed the available memory but also inflate the runtime of the model fitting. Therefore, fitting CWB with `mboost` to massive amounts of data is often not even feasible on modern workstations.

Chapter 6 introduces a software called `compboost` that provides an efficient implementation of CWB. Increasing the efficiency on the software side means using a faster core language, e.g., `C++`, concentrating on the basic parts required for training, and making use of modern and optimized software such as `OpenMP` (Dagum and Menon, 1998) for parallel computations or `Armadillo` (Sanderson and Curtin, 2016, 2018) for matrix operations. Benchmarks conducted in Chapter 7 compared, among others, `compboost` with `mboost` and showed a speedup of up to 4. Incorporating the adaptions introduced in Section 3.2 and 3.3 further increases that speedup up to a factor of 32.

3.2 Accelerated optimization

One of the possible methodological adaptations already mentioned is to use faster optimization routines. Nowadays, the palette of gradient descending methods ranges from simple gradient descent variants like stochastic gradient descent to methods aiming to accelerate the convergence, such as Adaptive Moment Estimation (ADAM; Kingma and Ba, 2014). Another of these methods is Nesterov's accelerated gradients (NAG; Nesterov, 1983) applied to gradient descent, also called Nesterov momentum. These methods, originally designed for optimization in parameter spaces, can be extended to functional gradient descent. Hence, gradient boosting can also benefit from NAG to accelerate the fitting process, as presented by Biau et al. (2019). Lu et al. (2020) outlined that the latter approach may diverge in certain situations. As a solution, they propose an improved algorithm called Accelerated Gradient Boosting Machine (AGBM).

Instead of updating only the primary model $\hat{f}$, as termed in AGBM, an additional momentum model $\hat{h}$ is fitted. The full updating scheme contains a third model sequence $\hat{g} = (1 - \vartheta_m)\hat{f} + \vartheta_m \hat{h}$ as convex combination of $\hat{f}$ and $\hat{h}$ with weight $\vartheta_m = (m + 1)^{-1}$. The primary model is updated based on that sequence via $\hat{f}^{[m]} = \hat{g}^{[m]} + \nu \hat{b}^{[m]}$. The base component $\hat{b}^{[m]}$ added to $\hat{f}^{[m]}$ is fitted to the pseudo residuals $\boldsymbol{r}^{[m]}$ w.r.t. to $\hat{g}^{[m]}$ instead of $\hat{f}^{[m]}$. Finally, a second base learner $\hat{b}_{\text{cor}}^{[m]}$ is fitted to so-called error-corrected pseudo residuals defined as $c^{[m](i)} = r^{[m](i)} + \frac{m}{m+1}(c^{[m-1](i)} - \hat{b}_{\text{cor}}^{[m-1]}(\mathbf{x}^{(i)}))$, $i = 1, \dots, n$, if $m > 1$ and $\boldsymbol{c}^{[m]} = \boldsymbol{r}^{[m]}$ if $m = 0$. These error-corrected pseudo residuals contain the information of all previous iterations and are used to update the momentum model $\hat{h}^{[m]} = \hat{h}^{[m-1]} + \gamma\nu/\vartheta_m \hat{b}_{\text{cor}}^{[m]}$ with momentum parameter $\gamma \in (0, 1]$. The acceleration of AGBM is due to the momentum sequence and the inclusion of the second base learner $\hat{b}_{\text{cor}}^{[m]}$ fit to $\boldsymbol{c}^{[m]}$ in each iteration. Nevertheless, fitting a second base learner doubles the computational complexity in each iteration. Hence, obtaining a runtime improvement requires accelerating the fitting by a factor of at least 2.

Applying the idea of AGBM to CWB is presented in Chapter 7 and assembled in a method called accelerated CWB (ACWB). The objective of ACWB is not to improve CWB's predictive performance, but rather to increase the runtime without sacrificing predictive performance. Because AGBM ensembles trees as base learners, the model updates are allocated as tree predictions. In contrast, ACWB also requires updating the model parameters in each iteration to estimate partial feature effects and maintain the advantages of CWB described in Section 2.2.3. Further, ACWB can perform worse than CWB in certain situations due to aggressive acceleration. To address this shortcoming, Chapter 7 also presents a second algorithm called hybrid CWB (HCWB). HCWB starts with a burn-in stage using ACWB for rapidly finding a good, but potentially not the best, model and then continues with CWB to fine-tune the model estimate. For HCWB, it is then possible to use a higher momentum value to benefit from greater acceleration in the beginning. At the same time, ACWB should be used with a smaller momentum value to not overshoot a good solution. As shown in benchmarks on six different data sets, ACWB and HCWB yield an average speedup of 3.5 and 2.4, despite fitting a second base learner in each iteration and without losing predictive power when compared to CWB.

3.3 More efficient data representations

In addition to runtime, the second aspect of computational complexity addressed in this thesis is the memory consumption of CWB. Two stages are distinguished for a better understanding of how CWB allocates memory.

Initialization – During initialization, all base learners compute data required to estimate parameters. This phase is executed before conducting Algorithm 1 and prepares all base learners for estimating parameters $\hat{\boldsymbol{\theta}}^{[m]}$. Hence, during initialization, the kth base learner stores the design matrix $\mathbf{Z}_k \in \mathbb{R}^{n \times d_k}$

with nd_k (numerical) values. Note that fitting CWB can be sped up by storing, e.g., the Cholesky decomposition of $\mathbf{Z}_k^\mathsf{T}\mathbf{Z}_k + \boldsymbol{K}_k$, since its inverse is required for calculating $\hat{\boldsymbol{\theta}}_k^{[m]}$ (cf. Equation (2.3)). However, calculating the inverse matrix is independent of iteration m, and hence, all parts for conducting that operation can be reused in each iteration. Thus, a matrix of size $d_k \times d_k$ may also be stored per base learner during the initialization phase. In this thesis, this pre-allocation is referred to as caching.

Fitting – As described in Algorithm 1, the fitting phase uses the objects created during the initialization and estimates $\hat{\boldsymbol{\theta}}^{[m]}$ in each iteration. Therefore, the stored data are the selected base learner $\hat{\boldsymbol{\theta}}^{[m]}$. Further, depending on the implementation, pseudo residuals $\boldsymbol{r}^{[m]}$ or the current model predictions $\hat{y}^{(i)} = \hat{f}^{[m]}(\mathbf{x}^{(i)})$, $i \in \{1, \dots, n\}$, may also be stored. It may be beneficial to store these objects for every iteration to run diagnostics after or during the training. However, storing additional data further increases memory consumption and can slow down the fitting process.

The following considers methods that aim to reduce the memory load of the base learners in CWB. Here, the focus is on numerical features, since categorical features can be stored very efficiently as integers. The handling of categorical features is shortly explained and should act as a first example of how alternative data representations are used to represent the full design matrix $\mathbf{Z}_k$ by a reduced representation.

A categorical feature representation – Usually, the design matrix $\mathbf{Z}_k$ of a base learner b_k modelling a categorical feature $x_k^{(i)} \in \{1, \dots, G\}$ with G classes uses a dummy or one-hot encoding (Tutz and Gertheiss, 2016). The one-hot encoding relies on basis functions $g(x_k) = (\mathbb{1}_{\{1\}}(x_k), \dots, \mathbb{1}_{\{G\}}(x_k))^\mathsf{T}$. Here, the categorical feature already contains information about the non-zero index and, hence, it is not necessary to fully build $\mathbf{Z}_k$. Thus, the one-hot encoding requires just storing n integer values as representative of $\mathbf{Z}_k$. Using a double-valued design matrix without a sparse matrix representation (cf. Section 3.3.1) requires storing Gn double values. Instead of applying Equation (2.3), which requires $\mathcal{O}(n^3)$ operations, a simple loop over $i = 1, \dots, n$ that updates $\hat{\theta}_{k,x_k^{(i)}} \leftarrow \hat{\theta}_{k,x_k^{(i)}} + n_{k,x_k^{(i)}}^{-1} y^{(i)}$ starting with $\hat{\theta}_{k,x_k^{(i)}} = 0$ and $n_{k,x_k^{(i)}}$ the number of observations associated with class $x_k^{(i)}$ merely requires $\mathcal{O}(n)$ operations to estimate the parameter vector $\hat{\boldsymbol{\theta}}_k$ as group means (if no penalty is applied).

3.3.1 Numerical feature representations

Sparse matrices – Using sparse matrices (see, e.g., Duff et al., 1989) can reduce the allocated memory of a design matrix $\mathbf{Z}_k$. The memory reduction depends on the sparsity or the number of non-zero elements of the matrix $\mathbf{Z}_k$. The goal of sparse data formats is to store the matrix dimensionality (n, d_k) and the non-zero elements $(v^{(1)}, \dots, v^{(n_k^*)})$ with $n_k^* < d_k n$ entries. The respective row and column locations $(i, j)^{(l)}$, $l = 1, \dots, n_k^*$, are stored as two integer vectors $(i^{(1)}, \dots, i^{(n_k^*)})$ and $(j^{(1)}, \dots, j^{(n_k^*)})$. The matrix is then fully defined by tuples $(i^{(l)}, j^{(l)}, v^{(l)})$, $l = 1, \dots, n_k^*$. This encoding is used for the coordinate list (COO; Shahnaz et al., 2005) format. Another widely used sparse matrix format, e.g., by the `R` package `Matrix` (Bates et al., 2022) or the `C++` library `Armadillo` (Sanderson and Curtin, 2016), is the compressed sparse column (CSC; Barrett et al., 1994, Section 4.3.1) format. The objective here is not to store each row and column index individually, but all row indices $(i^{(1)}, \dots, i^{(n_k^*)})$ and a vector, also referred to as a column pointer, $(j^{(1)}, j^{(2)}, \dots, j^{(d_k+1)})$, with $j^{(1)} = 0$, specifying where to split the value and row vectors to obtain the column entries. Column l is then obtained by values

$(v^{(j^{(l)}+1)}, \dots, v^{(j^{(l+1)})})$ and, respectively, rows $(i^{(j^{(l)}+1)}, \dots, i^{(j^{(l+1)})})$. All other entries of the matrix are set to zero. If the l^{th} column is empty, it is expressed by two equal values $j^{(l)} = j^{(l+1)}$. For example, a $n \times 5$ matrix with $n_k^* = 10$ and column pointer $(0, 2, 5, 5, 6, 10)$ defines the matrix according to:

column:	1		2			3	4	5			
values:	$v^{(1)}$,	$v^{(2)}$,	$v^{(3)}$,	$v^{(4)}$,	$v^{(5)}$,		$v^{(6)}$,	$v^{(7)}$,	$v^{(8)}$,	$v^{(9)}$,	$v^{(10)}$
row indices:	$i^{(1)}$,	$i^{(2)}$,	$i^{(3)}$,	$i^{(4)}$,	$i^{(5)}$,		$i^{(6)}$,	$i^{(7)}$,	$i^{(8)}$,	$i^{(9)}$,	$i^{(10)}$
column pointer:		2			5	5	6				10

Using sparse data representations not only reduces the required memory, but also gains faster matrix multiplications (see, e.g., Davis, 2006, Chapter 2.8), as efficiently accessing the columns is possible. An example of applying sparse matrices is representing the P-spline basis in a model. First, the memory is reduced, since each row usually contains $d_k^* < d_k$ non-zero elements. Second, using optimized matrix operations speeds up the fitting of the P-spline.

Binning numerical features – Sometimes in practice, when dealing with massive amounts of data, the fitting may crash due to the lack of memory – even when using an efficient categorical and sparse data representation. Lang et al. (2014) used binning to discretize feature vectors to increase the efficiency of multilevel structured additive regression. To bin a numerical feature $\mathbf{x}_k$ into n^* values, they used a simple grid of design points $\boldsymbol{z}_k = (z_k^{(1)}, \dots, z_k^{(n^*)})^\mathsf{T}$ with elements $z_k^{(i)} = \min(\mathbf{x}_k) + (i-1)/(n^*-1)(\max(\mathbf{x}_k) - \min(\mathbf{x}_k))$, $i = 1, \dots, n^* \in \mathbb{N}$. Binning is then accomplished by replacing each value $x_k^{(i)}$ with its closest design point $z_k^{(i)}$. The index vector $\mathbf{ind}_k = (\text{ind}_k^{(1)}, \dots, \text{ind}_k^{(n)})^\mathsf{T}$ represents this assignment with elements

$$\text{ind}_k^{(i)} = \begin{cases} 1 & x_k^{(i)} \in [z_k^{(1)}; z_k^{(2)} - m^{(1)}] \\ l \quad \text{if} & x_k^{(i)} \in (z_k^{(l)} - m^{(l-1)}; z_k^{(l)} + m^{(l)}] \\ n^* & x_k^{(i)} \in (z_k^{(n^*)} - m^{(n^*-1)}; z_k^{(n^*)}] \end{cases}$$

where $m^{(l)} = 0.5(z_k^{(l+1)} - z_k^{(l)})$ is half the distance of the l^{th} design point $z_k^{(l)}$ to its right neighbor $z_k^{(l+1)}$. The feature vector $\mathbf{x}_k$ is then represented by the n^* design points $\boldsymbol{z}_k$ and assignments $\mathbf{ind}_k$ with $x_k^{(i)} \approx z_k^{(\text{ind}_k^{(i)})}$. Wood et al. (2017) applied binning to fit GAMs to – as they call extremely large amounts of data – gigadata. They also argue that the best approximation is achieved by setting $n^* = \sqrt{n}$. Building on that, Li and Wood (2020) presented optimized cross-product operations of design matrices based on binned features. Besides the memory reduction, these operations further speed up the runtime.

Consequently, binning is also a suitable extension for CWB because it uses the same base representation as the additive terms in GAMs and an individual treatment of base learners. Treating base learners individually means that it is possible to use, e.g., sparse matrices to model P-splines for base learner b_j, while base learner b_k builds on a dense design matrix for polynomial regression. Treating both base learners differently does not affect one or the other because fitting CWB only relies on the contribution to the overall prediction $\hat{f}$ of one base learner at a time.

As shown in simulations outlined in Chapter 7, it is possible to save memory up to a factor of 5 for data sets with just numerical features. Further, binning in the pure numerical case can reduce the runtime by a factor of 4 - 6.

3.3.2 Example: Memory usage for different scenarios

The following example provides an intuition for the required memory during initialization in different scenarios. To that end, consider a data set $\mathcal{D}$ with p numerical and no categorical features. For each scenario, K design matrices $\mathbf{Z}_1, \dots, \mathbf{Z}_K$ (one per feature), pseudo residuals $\boldsymbol{r}^{[m]}$, and predictions $\hat{\mathbf{y}}^{[m]} = (\hat{f}^{[m]}(\mathbf{x}^{(1)}), \dots, \hat{f}^{[m]}(\mathbf{x}^{(n)}))^\mathsf{T}$ are stored. Further, integer values require i^{int} and double values i^{dbl} bytes. Hence, storing the pseudo residuals and predictions requires $2ni^{\text{dbl}}$ bytes. For simplicity, it is assumed that $d_k = d, \forall k \in \{1, \dots, K\}$, and each feature is modeled as a P-spline with d_k^* non-zero elements in each row of $\mathbf{Z}_k$. Further, sparse matrices use the COO format with two integer-sized vectors $d_k^* n$ and one double-sized vector storing $d_k^* n$ elements. Hence, $d_k^* n i^{\text{dbl}} + 2 d_k^* n i^{\text{int}}$ bytes are allocated per sparse design matrix. Table 3.1 contains the estimated memory consumption for different scenarios. These estimates are calculated by setting[1] $i^{\text{int}} = 4$, $i^{\text{dbl}} = 8$, $K = p$, $n^* = \sqrt{n}$ and $d = 24$ with $d_k^* = 5$. The scenarios considered are:

(1) Each design matrix $\mathbf{Z}_k$ is stored as dense matrix, no caching:

$$\text{mem}_1(n,p) = Kndi^{\text{dbl}} + 2ni^{\text{dbl}} \text{ bytes} \overset{\text{values}}{=} 192pn + 16n \text{ bytes}$$

(2) Each design matrix $\mathbf{Z}_k$ is stored as dense matrix, with caching:

$$\text{mem}_2(n,p) = Kndi^{\text{dbl}} + Kd^2 i^{\text{dbl}} + 2ni^{\text{dbl}} \text{ bytes} \overset{\text{values}}{=} 192pn + 4608p + 16n \text{ bytes}$$

(3) Each design matrix $\mathbf{Z}_k$ is stored as sparse matrix, with caching:

$$\begin{aligned}\text{mem}_3(n,p) &= K(d_k^* n i^{\text{dbl}} + 2d_k^* n i^{\text{int}}) + Kd^2 i^{\text{dbl}} + 2ni^{\text{dbl}} \text{ bytes} \\ &\overset{\text{values}}{=} 80pn + 4608p + 16n \text{ bytes}\end{aligned}$$

(4) Each design matrix $\mathbf{Z}_k$ is stored as sparse matrix with binning, with caching:

$$\begin{aligned}\text{mem}_4(n,p) &= K(d_k^* n^* i^{\text{dbl}} + 2d_k^* n^* i^{\text{int}} + ni^{\text{int}}) + Kd^2 i^{\text{dbl}} + 2ni^{\text{dbl}} \text{ bytes} \\ &\overset{\text{values}}{=} p(80\sqrt{n} + 4n) + 4608p + 16n \text{ bytes}\end{aligned}$$

Table 3.1 contains the memory consumption for these four scenarios with $n \in \{100, 10\,000, 1\,000\,000\}$ and $p \in \{10, 50, 100, 500\}$. Using caching (difference between (1) and (2)) has a relevant impact for small n and large p. The difference between using sparse matrices over dense matrices (difference between (2) and (3)) has a more relevant impact, whereas binning helps to reduce the allocated memory further. It is noteworthy that these examples are calculated based on the minimal possible amount of data needed to be stored during initialization. For $n = 1\,000\,000$ and $p = 500$, a machine must allocate about 40 GB of RAM for scenario (3), while it is just 2 GB for scenario (4). This difference can make it

[1]This parameter setting is an often-used default for P-splines with cubic splines, second order penalty differences, and 20 knots.

feasible to fit CWB on notebooks instead of using more powerful workstations or servers. Implementations often save additional information, such as parameter estimates or the pseudo residuals/predictions for each iteration. Hence, these numbers cannot be transferred one-to-one to real-world applications. Nevertheless, it illustrates the effectiveness of efficient data representations. A comparison of different data representations on simulated data using `compboost` reveals memory savings of up to a factor of 5 (cf. Chapter 7). To underpin the effectiveness of binning, Chapter 7) also shows three larger example data sets ($\approx$ 3 GB) for which fitting CWB is not possible without binning.

	$p = 10$	$p = 50$	$p = 100$	$p = 500$
$n = 100$	(1): 0.18 MB (1.52) (2): 0.23 MB (1.88) (3): 0.12 MB (1) (4): 0.06 MB (0.47)	(1): 0.92 MB (1.52) (2): 1.14 MB (1.89) (3): 0.6 MB (1) (4): 0.28 MB (0.46)	(1): 1.83 MB (1.52) (2): 2.27 MB (1.89) (3): 1.2 MB (1) (4): 0.56 MB (0.46)	(1): 9.16 MB (1.52) (2): 11.35 MB (1.89) (3): 6.01 MB (1) (4): 2.77 MB (0.46)
$n = 10\,000$	(1): 18.46 MB (2.36) (2): 18.51 MB (2.36) (3): 7.83 MB (1) (4): 0.65 MB (0.08)	(1): 91.71 MB (2.38) (2): 91.93 MB (2.39) (3): 38.52 MB (1) (4): 2.66 MB (0.07)	(1): 183.26 MB (2.38) (2): 183.7 MB (2.39) (3): 76.89 MB (1) (4): 5.17 MB (0.07)	(1): 915.68 MB (2.39) (2): 917.88 MB (2.39) (3): 383.82 MB (1) (4): 25.24 MB (0.07)
$n = 1\,000\,000$	(1): 1846.31 MB (2.37) (2): 1846.36 MB (2.37) (3): 778.24 MB (1) (4): 54.21 MB (0.07)	(1): 9170.53 MB (2.39) (2): 9170.75 MB (2.39) (3): 3830.18 MB (1) (4): 210.03 MB (0.05)	(1): 18325.81 MB (2.4) (2): 18326.25 MB (2.4) (3): 7645.09 MB (1) (4): 404.8 MB (0.05)	(1): 91567.99 MB (2.4) (2): 91570.19 MB (2.4) (3): 38164.43 MB (1) (4): 1962.95 MB (0.05)

Table 3.1: Memory consumption for different scenarios (1) to (4). The number in brackets after the allocated MB is the relative memory consumption when compared to scenario (3). Scenario (3) is used as reference since `mboost`, as well as `compboost`, applies it as default. For example, having $n = 10\,000$ and $p = 100$ (scenario (2)) requires 2.39 times more memory than scenario (3). The scenario with binning requires just 7 % of the memory, or about 14 times less memory than scenario (3).

CHAPTER 4

Automation

Nowadays, more people, firms, and institutions show a steadily growing interest in data analytics, where easy access to ML algorithms attracts them to utilize ML to gain value from their data. However, not all users are experts in the field of ML, and many do not understand the algorithmic details to satisfyingly develop a bespoke model for their needs. AutoML (Hutter et al., 2019) aims to automate preprocessing, model selection, and postprocessing by solving the Combined Algorithm and Hyperparameter Selection (CASH) problem (Thornton et al., 2013). In practice, users must choose an AutoML framework, the objective or performance measure to optimize, set constraints for the optimization, and finally feed the framework with a data set. To that end, all pre- and postprocessing steps and supported model classes of an AutoML framework are defined by experts, i.e., the developer of the framework, and optimized with HPO. Existing AutoML frameworks are, e.g., Auto-WEKA (Kotthoff et al., 2017), Auto-sklearn (Feurer et al., 2015), or autoxgboost (Thomas et al., 2018). These frameworks are able to achieve high predictive performance due to using black box models and optimizing for the best objective. However, in most cases, AutoML does not generally lend itself to the generation of highly interpretable or explainable[1] models.

In the following, ML pipelines (Section 4.1) and the trade-off between model performance and interpretability (Section 4.2) are briefly discussed. Section 4.3 argues for the need for AutoML systems that promote interpretability. Building on this chapter, Chapter 8 introduces a prototype for an AutoML framework called `Autocompboost` that is based on CWB and yields fully explainable models.

4.1 Machine learning pipelines

Using data requires handling, e.g., missing values (Van Buuren, 2018) or outliers (John, 1995), but grouping levels of categorical features or accounting for underrepresented classes may also be necessary. In this thesis, the data set $\mathcal{D}$ is a result of preprocessing raw data $\mathcal{D}_r$ and the input to the inducer to fit a model $\hat{f} = \mathcal{I}_{\boldsymbol{\lambda}}(\mathcal{D})$ with HPs $\boldsymbol{\lambda}$. Hence, a preprocessing operator f_{op} transforms the raw data set $\mathcal{D}_r$ into the training data $\mathcal{D} = f_{\text{op}}(\mathcal{D}_r|\boldsymbol{\theta}_{\text{op}})$. The operator f_{op} is defined by parameters $\boldsymbol{\theta}_{\text{op}}$ and can be trained as result of a process $\hat{f}_{\text{op}} = \mathcal{I}_{\text{op},\boldsymbol{\lambda}_{\text{op}}}(\mathcal{D}_r)$ with additional HPs $\boldsymbol{\lambda}_{\text{op}} \in \Lambda_{\text{op}}$. To provide an example, during imputation with the mean, the average of the j^{th} feature in the raw data set $\mathcal{D}_r$ is stored in $\hat{\boldsymbol{\theta}}_{\text{op}}$ as the result of $\mathcal{I}_{\text{op},\boldsymbol{\lambda}_{\text{op}}}$ and is used to prepare new data sets for training. Therefore, a new raw data set $\mathcal{D}_r^*$ is transformed to $\mathcal{D}^* = \hat{f}_{\text{op}}(\mathcal{D}_r^*|\hat{\boldsymbol{\theta}}_{\text{op}})$ based on specific rules defined by $\hat{\boldsymbol{\theta}}_{\text{op}}$, such as imputing missing values with the mean of the raw data set $\mathcal{D}_r$.

[1]Throughout this work, interpretable and explainable are treated as exchangeable synonyms.

Further, multiple preprocessing operators $f_{\text{op},1}, \dots, f_{\text{op},K}$ can be composed to one operator $f_{\text{op}} = f_{\text{op},1} \circ \cdots \circ f_{\text{op},K}$. For example, detecting outliers with $f_{\text{op},1}$, grouping categorical features with $f_{\text{op},2}$, and using a principal component analysis for feature extraction with $f_{\text{op},3}$ are assembled to give $f_{\text{op}} = f_{\text{op},1} \circ f_{\text{op},2} \circ f_{\text{op},3}$. Each operator $f_{\text{op},k}$ is attached with a parameter vector $\hat{\boldsymbol{\theta}}_{\text{op},k}$, and therefore, f_{op} is defined by $\hat{\boldsymbol{\theta}}_{\text{op}} = \{\hat{\boldsymbol{\theta}}_{\text{op},1}, \dots, \hat{\boldsymbol{\theta}}_{\text{op},K}\}$. Respectively, the HPs of f_{op} are $\boldsymbol{\lambda}_{\text{op}} = \{\boldsymbol{\lambda}_{\text{op},1}, \dots, \boldsymbol{\lambda}_{\text{op},K}\}$ with $\boldsymbol{\lambda}_{\text{op},k} \in \Lambda_{\text{op},k}$. Transforming raw data according to $\hat{f}_{\text{op}}$ is especially crucial during evaluation when the data set is repeatedly split into several data subsets for training and testing (cf. Section 2.3).

Hence, the ML pipeline is the arrangement of the preprocessing and model fitting as nodes in a linear graph (cf., e.g., Bischl et al., 2021, Section 5.1)[2], starting with fitting $\hat{f}_{\text{op}} = \mathcal{I}_{\text{op},\boldsymbol{\lambda}_{\text{op}}}(\mathcal{D}_r)$ and finishing with fitting $\hat{f} = (\mathcal{I}_{\boldsymbol{\lambda}} \circ \hat{f}_{\text{op}})(\mathcal{D}_r)$. Note that the whole process for predicting new data $\mathcal{D}_r^*$ requires passing $\mathcal{D}^*$ through the whole trained pipeline, starting with $\mathcal{D}^* = \hat{f}_{\text{op}}(\mathcal{D}_r^* | \hat{\boldsymbol{\theta}}_{\text{op}})$ and calculating predictions with $\hat{f}(\mathbf{x}^*),\ \forall \mathbf{x}^* \in \mathcal{D}^*$. Formalizing that process allows conducting HPO to find a good performing configuration for $\{\boldsymbol{\lambda}_{\text{op}}, \boldsymbol{\lambda}\}$ and requires optimizing over a complex space $\Lambda_{\text{op},1} \times \cdots \times \Lambda_{\text{op},K} \times \Lambda$. Because of that complex HP space, optimizing an ML pipeline is tough and requires black box optimization (Bischl et al., 2021). Therefore, fitting AutoML systems is a computationally demanding task.

4.2 Accuracy–interpretability trade-off

As described above, AutoML frameworks often aggressively optimize an ML pipeline to achieve high predictive performance. A cost for obtaining high predictive performance is to use black box models that are not interpretable. This can create tension between accuracy and explainability (Freitas, 2019; Drozdal et al., 2020; Xanthopoulos et al., 2020) and is often referred to as the accuracy–interpretability trade-off. However, accurate prediction is often not the primary goal. Thus, high-performing non-interpretable frameworks are not applicable for practitioners who solely aim for explaining their model. Explainability is especially important in fields with a high impact on people's lives, such as health, finance, insurance, or education. Therefore, to solve that problem, ongoing research focuses on using black box models and their high predictive power in combination with making them interpretable by postprocessing using IML (Molnar, 2020) also referred to as XAI (Arrieta et al., 2020). These methods aim to recover interpretability by, e.g., decomposing the model prediction (Ribeiro et al., 2016), estimating partial feature effects (see, e.g., Molnar, 2020, Chapter 8.1), or evaluating feature importance (Lundberg and Lee, 2017). Thus, combining black box models with IML can circumvent the accuracy–interpretability problem by fitting high-performing models and relying on post-hoc explanations. Nevertheless, regardless of the advantages of IML, Rudin (2019) assert that explaining black box models is not reliable and can harm society and effort should instead be spent on using inherently interpretable models. The author argues that often not enough details are provided to understand what the black box is doing or that explanations can become overly complicated and, hence, are prone to human errors.

[2]ML pipelines can also account for postprocessing operations, e.g., calibrating probabilities or threshold tuning (cf., Bischl et al., 2021, Section 4.5), by adding a postprocessing operator which is not further discussed in this work. For additional details and explanations about the concept of ML pipelines see, e.g., Pfisterer (2022).

4.3 Trustworthy automated machine learning

As described, AutoML is a valuable tool to help non-expert practitioners create proper models for their data by optimizing ML pipelines. Nevertheless, if interpretability is desired, these models often require techniques from IML to understand their behavior. But, adding a potentially error-prone layer of complexity requires knowledge about these methods and trust that explanations gained from IML adequately capture the underlying processes. Estimating the future model performance is usually included in the evaluation during HPO, but generating trust in the interpretability capabilities of an AutoML framework should be treated as equally important. Hence, it is beneficial to know whether interpretable models are sufficient for modeling a given task. Notably, some AutoML systems, e.g., Auto-WEKA, optimize multiple models from simple interpretable ones (such as (generalized) LMs) to more complex ones (such as Random Forests (Breiman, 2001) or AdaBoost (Freund and Schapire, 1997)) and output the performance. This performance can indicate whether it is worth using a black box model over a simpler interpretable model to gain a satisfying level of performance. However, fitting many models is computationally demanding, and the increase in complexity is due to automatically changing the model class by the framework without letting the user control for complexity.

Thus, slightly increasing complexity and assessing the predictive performance for each increase gives a finer resolution of the required complexity for a given task. For example, tracing the risk improvement $\mathcal{R}_{\text{emp}}(\hat{f}^{[m]}|\mathcal{D}) - \mathcal{R}_{\text{emp}}(\hat{f}^{[m-1]}|\mathcal{D})$ in each iteration when using boosting can act as a proxy for the importance of a new component $\hat{b}^{[m]}$. Further, in each iteration, the new component $\hat{b}^{[m]}$ may add more complexity to the model. When using, e.g., CWB as a fitting engine, complexity can be increased by including a new feature, switching from linear to non-linear effects, including new pairwise interactions, or even boosting decision trees (Friedman, 2001) – which then enables accounting for higher-order interactions. To that end, the importance of the complexity added by $\hat{b}^{[m]}$ can be quantified w.r.t. to predictive performance. Hence, assessing each step and preparing the information in a readable and intuitive manner helps the user to understand the fitting procedure and indicates how cautiously interpretable base models can be used. Further, if interpretable models are not sufficient to achieve a desired predictive performance, it is reasonable to switch to performance-based AutoML frameworks. Chapter 8 introduces `Autocompboost` as an example of an AutoML framework that provides a complexity assessment based on CWB. `Autocompboost` further combines interpretable base components with trees to represent the full range of complexity from simple models – e.g., by just using univariate linear feature effects – to complex black box models by adding trees. This range allows assessing the gain in performance with increasing complexity on a fine grid and can improve practitioners' trust in the returned model.

CHAPTER 5

Distributed computing

The last extension of CWB presented in this thesis is to add the ability to fit CWB to distributed data. As described in Section 2.4, the distributed data sets $\mathcal{D}_1, \dots, \mathcal{D}_S$ are shared across S sites and exclusively located there. When analyzing data, merging the different data sets into a pooled data set $\mathcal{D} = \cup_{s=1}^{S} \mathcal{D}_s$ is strictly prohibited. A practical reason for this prohibition could be data security to protect the privacy of individuals in a data set. A prominent real-world scenario of distributed data sets is the collaboration of hospitals to understand the behavior of their patients for a specific treatment. Here, sharing parts of the data infringes on patient privacy and is legally forbidden. This need for security gave rise to privacy-preserving techniques such as federated learning (McMahan et al., 2017), the integration of encryption (Bost et al., 2014; Sun et al., 2020; Fang and Qian, 2021), or DP in data analysis (Abadi et al., 2016; Jayaraman and Evans, 2019; Gong et al., 2020).

The novel distributed techniques presented in this thesis were developed in collaboration with the Data Integration for Future Medicine (DIFUTURE) consortium (Prasser et al., 2018) as part of the German Medical Informatics Initiative[1] (MII). The consortium aims to provide digital tools for individual treatment decisions and prognosis and conduct distributed data network studies. A use case is to develop a score to understand the treatment of multiple sclerosis patients. These patients are located at different hospitals that are part of DIFUTURE. Specific challenges for that use case are the model development and evaluation in a distributed fashion. In the following, Section 5.1 gives more details about model fitting in a distributed context. Subsequently, Section 5.2 introduces model evaluation in the context of distributed data. Throughout this section, the medical context is used as an example to motivate different aspects of distributed analyses.

5.1 Model fitting

A challenge when fitting an algorithm to distributed data is to incorporate security mechanisms – e.g., data aggregation, DP, or encryption – into the fitting process so that the final model can be accurately estimated, ideally in a lossless manner. However, this might not be possible for complex algorithms. For example, Jayaraman and Evans (2019) studied the effect of the privacy parameters (ε, δ) in DP on the accuracy of different models and argued that meaningful privacy must be calibrated for each algorithm independently. Hence, defining an algorithm as privacy-preserving means accounting for privacy using one or multiple privacy mechanisms. Privacy-preserving algorithms in distributed computing ranges

[1] www.medizininformatik-initiative.de

from simple LMs (Karr et al., 2005) or ridge regression (Chen et al., 2018) to trees, support vector machines, random forests (Li et al., 2020), AdaBoost (Lazarevic and Obradovic, 2001; Gambs et al., 2007), or neural networks (Mohassel and Zhang, 2017).

However, finding a privacy-preserving alternative of an algorithm may not be sufficient to adequately model distributed data (Cunha et al., 2021). As set up in Section 2.4, the data sets $\mathcal{D}_s$ are heterogeneous, and this heterogeneity is induced by potentially different data distributions $\mathbb{P}_{xy,s}$. Hence, a distributed algorithm should be able to account for these different distributions. For example, the population of patients treated in one hospital can be substantially different from those at another hospital. Reasons can be, for example, regional aspects (hospitals are in different countries or urban vs. rural regions), the type of hospital (private or university), or different devices used to measure medical indicators. It may be reasonable to consider fitting different models $\hat{f}_s$ at each site s to account for heterogeneities. However, learning common patterns as main effects between all sites is beneficial for several reasons. New sites that did not contribute to the training can use the model with the main effects for predicting their data. Further, a feature effect may already be fully explained by a common main effect and does not require site-specific corrections. In this scenario, adding the site-specific effect is unnecessary to keep the model as simple as possible. Thus, a model should be able to model main effects that are common between all sites as well as site-specific correction for the sites.

Furthermore, especially in medicine, e.g., when selecting the treatment for a patient based on a model, understanding the decision-making of that model is critical. Hence, accessing the interpretations for the model is often desired. Therefore, this thesis deals with adjusting CWB to make it privacy-preserving and account for heterogeneities. Thus, the model given in Equation (2.1) is extended to

$$g(\mathbb{E}[Y|\mathbf{x}, s]) = f(\mathbf{x}) = f_0 + \sum_{k=1}^{K} \left(b_k(\mathbf{x}|\boldsymbol{\theta}_k) + \sum_{j=1}^{S} \mathbb{1}_{\{s\}}(j) b_{k,j}(\mathbf{x}|\boldsymbol{\theta}_{k,j}) \right) \tag{5.1}$$

with site-specific terms $b_{k,s}$, $s = 1, \dots, S$ to account for different data distributions and indicator function $\mathbb{1}_{\{s\}}(j) = 1$ if $j = s$ and 0 otherwise. Another view on this decomposition is to express it as a GAMM. Thus, the effect of base learner b_k corresponds to the fixed effect of feature k while $b_{k,s}$ is a smooth random effect of feature k to account for site s that acts as a statistical unit. To that end, the repeated measurements are given by $\mathcal{D}_s$. Zhu et al. (2020), Luo et al. (2022), and Yan et al. (2022) tackle the issue of data heterogeneity by using a linear random component to estimate a privacy-preserving and lossless (general) linear mixed model ((G)LMM). Nevertheless, the linear component may need to be more flexible for adequate modeling.

Chapter 9 describes how CWB can be used as a fitting engine for GAMMs in a distributed fashion for estimating (5.1). For the distributed CWB algorithm, it is sufficient to rely on aggregations similar to the distributed LM (Algorithm 2) for a distributed base learner estimation. The site-specific effects are included as row-wise tensor products to account for heterogeneities. Further, the presented distributed CWB algorithm preserves all advantages of CWB and thus allows a selection between the fixed and random effects. For example, suppose a fixed effect is sufficient for modeling, and no site-specific correction is required. In that case, the selection properties of the distributed CWB algorithm account for only selecting the fixed effect and ignoring the random component.

5.2 Model evaluation

The principles from model evaluation in non-distributed systems (Section 2.3) also apply to distributed data sets and become more complicated for several reasons. Data sets are distributed, so conducting resampling and performance evaluation of distributed data must comply with all safety mechanisms. For example, sharing prediction scores can already be enough to reconstruct training data (Wang and Kurz, 2022). On the other hand, sharing $\tilde{\rho}(\hat{f}, \mathcal{D}_s)$ is usually allowed if n_s is larger than or equal to the privacy level q (cf. Section 2.4.2) and can be used to calculate performance measures that are based on a point-wise loss function L. For example, calculating the mean squared error (MSE) – which is based on point-wise computing the L_2-loss in a distributed fashion – is done by sharing $l_s = \sum_{(\mathbf{x},y)\in\mathcal{D}_s} L_2(y, \hat{f}(\mathbf{x}))$ and calculating $\text{MSE}(\hat{f}, \mathcal{D}) = n^{-1}\sum_{s=1}^{S} l_s$ with $n = \sum_{s=1}^{S} n_s$. In general, models can be evaluated on distributed data sets by calculating the performance measure $\tilde{\rho}(\hat{f}, \mathcal{D}_s)$ per site and averaging all site performance measures. Nevertheless, for some measures, this does not reliably estimate the global performance and can be biased towards single sites. A reason could be that the calculation of the global performance measure is not decomposable into point-wise operations, or one site requires specific information from other sites – e.g., the AUC requires all scores and true outcomes from all sites.

Hence, a measure for which the evaluation with distributed data sets becomes even more complicated is the AUC. In contrast to point-wise losses, calculating the AUC requires processing the whole data set. In the case of the AUC, it is necessary to merge all scores and the true 0-1-outcomes to calculate global ranks and respective TPR and FPR values. Relying on aggregations is insufficient for the AUC, and special treatment is required for a distributed calculation. Chapter 10 presents a distributed version to calculate the ROC curve, AUC, and confidence intervals for the AUC. The estimation is based on approximating the ROC curve with the ROC-GLM (Pepe, 2003). Calculating the ROC-GLM distributively and preserving privacy is accomplished by incorporating data aggregations and DP. In general, distributed model evaluation must still be further researched. However, conducting an evaluation using performance measures based on point-wise loss functions can be securely conducted using data aggregations, while more complex loss functions, such as the AUC, require special treatment.

Part II - Contributions

CHAPTER 6

`compboost`: Modular Framework for Component-Wise Boosting

Contributing article

Schalk, D., Thomas, J., and Bischl, B. (2018). compboost: Modular framework for component-wise boosting. *Journal of Open Source Software*, 3(30):967

Declaration of contributions

Daniel Schalk worked with Janek Thomas on the underlying software design. Furthermore, Daniel Schalk implemented the method, created the first benchmark, and wrote the central part of the manuscript.

Contribution of the coauthors

Janek Thomas was heavily involved in the initial software design as well as the creation of the R-API. All co-authors helped revise the manuscript.

Note:

The publication followed the master thesis[1] of Daniel Schalk. The development of the R package is an ongoing process. In the master thesis, the first stable version was developed, which contained the basic algorithm. The functional scope of this first version included three loss functions (absolute, quadratic, and binomial loss) as well as the possibility to define own loss functions, two basic components for effect modeling (linear and P-spline), and an optimization method. After the master thesis, this functionality was extended. New loss functions (Huber and Poisson loss) and basic components (dummy coding, centered splines, tensor products) were added. In addition, the API was extended to provide new visualization capabilities that facilitate the interpretation of the model. Fundamentally, the internal structures have also been revised to allow more efficient data processing using sparse file formats. According to the JOSS publication, the software was extended to include parallelized model estimation and additional optimizers and base components. In 2019, compboost was also presented at UseR in Toulouse[2].

[1] `https://epub.ub.uni-muenchen.de/59109/1/MA_Schalk.pdf`
[2] `https://youtube.com/watch?v=nOMSQJU51Tk&ab_channel=RConsortium`

compboost: Modular Framework for Component-Wise Boosting

Daniel Schalk[1], **Janek Thomas**[1], **and Bernd Bischl**[1]

1 Department of Statistics, LMU Munich

DOI: 10.21105/joss.00967

Software

- Review
- Repository
- Archive

Submitted: 14 August 2018
Published: 12 October 2018

Summary

In high-dimensional prediction problems, especially in the $p \gg n$ situation, feature selection is an essential tool. A fundamental method for problems of this type is component-wise gradient boosting, which automatically selects from a pool of base learners – e.g. simple linear effects or component-wise smoothing splines (Schmid & Hothorn, 2008) – and produces a sparse additive statistical model. Boosting these kinds of models maintains interpretability and enables unbiased model selection in high-dimensional feature spaces (Hofner, Hothorn, Kneib, & Schmid, 2012).

The `R` (Team, 2016) package `compboost`, which is actively developed on GitHub (https://github.com/schalkdaniel/compboost), implements component-wise boosting in `C++` using `Rcpp` (Eddelbuettel, 2013) and `Armadillo` (Sanderson & Curtin, 2016) to achieve efficient runtime behavior and full memory control. It provides a modular object-oriented system which can be extended with new base-learners, loss functions, optimization strategies, and stopping criteria, either in `R` for convenient prototyping or directly in `C++` for optimized speed. The latter extensions can be added at runtime, without recompiling the whole framework. This allows researchers to easily implement more specialized base-learners, e.g., for spatial or random effects, used in their respective research area.

Visualization of selected effects, efficient adjustment of the number of iterations, and traces of selected base-learners and losses to obtain information about feature importance are supported.

Compared to the reference implementation for component-wise gradient boosting in `R`, `mboost` (Hothorn, Buehlmann, Kneib, Schmid, & Hofner, 2017), `compboost` is optimized for larger datasets and easier to extend, even though it currently lacks some of the large functionality `mboost` provides. A detailed benchmark against `mboost` can be viewed on the project homepage and on GitHub.

The modular design of `compboost` allows extension to more complicated settings like functional data or survival analysis. Further work on the package should include parallelized boosting, better feature selection, faster optimization techniques such as momentum and adaptive learning rates, as well as better overfitting control.

References

Eddelbuettel, D. (2013). *Seamless r and c++ integration with rcpp*. Springer. doi:10.1007/978-1-4614-6868-4

Hofner, B., Hothorn, T., Kneib, T., & Schmid, M. (2012). A framework for unbiased model selection based on boosting. *Journal of Computational and Graphical Statistics*, *20*(4), 956–971. doi:10.1198/jcgs.2011.09220

Hothorn, T., Buehlmann, P., Kneib, T., Schmid, M., & Hofner, B. (2017). *mboost: Model-based boosting*. Retrieved from https://CRAN.R-project.org/package=mboost

Sanderson, C., & Curtin, R. (2016). Armadillo: A template-based c++ library for linear algebra. *Journal of Open Source Software*, *1*(2), 26. doi:10.21105/joss.00026

Schmid, M., & Hothorn, T. (2008). Boosting additive models using component-wise p-splines. *Computational Statistics & Data Analysis*, *53*(2), 298–311.

Team, R. C. (2016). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

CHAPTER 7

Accelerated Componentwise Gradient Boosting Using Efficient Data Representation and Momentum-Based Optimization

Contributing article

Schalk, D., Bischl, B., and Rügamer, D. (2022a). Accelerated componentwise gradient boosting using efficient data representation and momentum-based optimization. *Journal of Computational and Graphical Statistics*

Declaration of contributions

Daniel Schalk transferred the methodological concepts of the improved optimization algorithm and the more efficient data representation to CWB. For this purpose, he constructed the essential theory and formulated the algorithms. The adaptations in the software `compboost`, the simulation study, and the comparison to state-of-the-art methods were also implemented solely by him. Daniel Schalk formulated main parts of the publication, interpreted the results, and created all figures.

Contribution of the coauthors

David Rügamer assisted in interpreting the theoretical and methodological findings as well as the results of the simulation study and benchmark comparison. Bernd Bischl and David Rügamer helped with editing the manuscript.

Accelerated Componentwise Gradient Boosting using Efficient Data Representation and Momentum-based Optimization

Daniel Schalk, Bernd Bischl and David Rügamer
Department of Statistics, LMU Munich

Abstract

Componentwise boosting (CWB), also known as model-based boosting, is a variant of gradient boosting that builds on additive models as base learners to ensure interpretability. CWB is thus often used in research areas where models are employed as tools to explain relationships in data. One downside of CWB is its computational complexity in terms of memory and runtime. In this paper, we propose two techniques to overcome these issues without losing the properties of CWB: feature discretization of numerical features and incorporating Nesterov momentum into functional gradient descent. As the latter can be prone to early overfitting, we also propose a hybrid approach that prevents a possibly diverging gradient descent routine while ensuring faster convergence. Our adaptions improve vanilla CWB by reducing memory consumption and speeding up the computation time per iteration (through feature discretization) while also enabling CWB learn faster and hence to require fewer iterations in total using momentum. We perform extensive benchmarks on multiple simulated and real-world data sets to demonstrate the improvements in runtime and memory consumption while maintaining state-of-the-art estimation and prediction performance.

Keywords: Binning, Data Structures, Functional Gradient Descent, Machine Learning, Nesterov Momentum

1 Introduction

Model-based or *componentwise boosting* (CWB; Bühlmann and Yu, 2003) applies gradient boosting (Freund et al., 1996) to statistical models by sequentially adding pre-defined components to the model. These components are so-called base learners of one or multiple features. If interpretable base learners are used (e.g., univariate splines), the full CWB model remains interpretable and allows for the direct assessment of estimated partial feature effects. Further advantages of CWB are its applicability in high-dimensional feature spaces ("$p \gg n$ situations"), its inherent variable selection, and unbiased feature selection (Bühlmann and Yu, 2003; Hofner et al., 2011). It is also possible to derive inference properties of boosted estimators to quantify uncertainty using post-selection inference procedures (Rügamer and Greven, 2020). These properties make CWB a powerful method at the intersection of (explainable) statistical modelling and (black-box prediction) machine learning. For this reason, CWB is frequently used in medical research, e.g., for oral cancer prediction (Saintigny et al., 2011), detection of synchronization in bioelectrical signals (Rügamer et al., 2018), or classifying pain syndromes (Liew et al., 2020). In contrast, many other gradient boosting methods such as XGBoost (Chen and Guestrin, 2016) solely focus on predictive performance and (mainly) use tree-based base learners with higher-order interactions. As a consequence, these procedures require techniques from interpretable machine learning (see, e.g., Molnar, 2020) to explain their resulting predictions.

Various versions of the original CWB algorithm have been developed, e.g., CWB for functional data (Brockhaus et al., 2020), boosting location, scale and shape models (Hofner et al., 2016), or probing for sparse and fast variable selection (Thomas et al., 2017). CWB's computational complexity in terms of memory and runtime is, however, a downside often encountered in practice. The high consumption of RAM of the current state-of-the-art implementation `mboost` (Hothorn et al., 2020) can considerably exceed the capacity of modern workstations. This makes CWB less attractive or even infeasible for medium- to large-scale applications. In this paper, we focus on two internal structures of CWB and propose improvements to mitigate these problems.

Our contributions and related literature. Our first contribution (Section 3.1) is a novel CWB modification to reduce memory consumption in large data situations. To the best of our knowledge, we are the first to derive the complexity of CWB and also the first to suggest improvements to reduce computational costs. Based on a recently proposed idea to fit generalized additive models (GAMs) on large data sets (Li and Wood, 2020), we describe adaptions of matrix operations for CWB to operate on discretized features. We refer to this approach as binning. Binning is a discretization technique for numerical features and can drastically reduce runtime and memory consumption when fitting GAMs, especially when coupled with specialized matrix operations. In contrast to Li and Wood (2020), we use binning within each base learner rather than processing the model matrix of all features. This makes the use of binning particularly beneficial for CWB.

In Section 3.2, we further adapt a novel optimization technique called Accelerated Gradient Boosting Machine (AGBM; Lu et al., 2020). AGBM allows incorporation of Nesterov momentum for gradient boosting in the function space. Based on AGBM, we propose a new variant of CWB for faster convergence while still preserving CWB's interpretability. The adaption perfectly fits to the general optimization scheme of CWB. However, the acceleration based on Nesterov momentum is known to diverge in certain cases (Wang et al., 2020). We thus also propose a refinement of AGBM in our algorithm Hybrid CWB (HCWB) to overcome premature divergence of the gradient descent routine.

These proposed adaptions can be applied to CWB independently of each other and are both implemented in the software package `compboost` (Schalk et al., 2018). In a simulation study in Section 4, we study their effect both separately and combined. Finally, we conduct benchmark experiments in Section 4 to compare our proposed CWB variants with vanilla CWB, the CWB implementation `mboost`, XGBoost, and the recently published Explainable Boosting Machine (EBM) within the `interpretML` framework (Nori et al., 2019).

2 Componentwise Gradient Boosting

This section introduces the main concepts and properties of CWB as well as our notation.

2.1 Terminology

Consider a p-dimensional feature space $\mathcal{X} = (\mathcal{X}_1 \times \ldots \times \mathcal{X}_p)$ and a target space $\mathcal{Y}$. We assume an unknown functional relationship f between $\mathcal{X}$ and $\mathcal{Y}$. ML algorithms try to learn this relationship using a data set $\mathcal{D} = \left(\left(\boldsymbol{x}^{(1)}, y^{(1)}\right), \ldots, \left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right)$ with n observations drawn independently from an unknown probability distribution $\mathbb{P}_{xy}$ on the joint space $\mathcal{X} \times \mathcal{Y}$. Let $\hat{f}$ be the estimated model fitted on the training data to approximate f and $P = \{1, \ldots, p\}$ an index set for all p features. The vector $\boldsymbol{x}_j = (x_j^{(1)}, \ldots, x_j^{(n)})^\mathsf{T} \in \mathcal{X}$ refers to the jth feature. $\boldsymbol{x}$ and y are arbitrary members of $\mathcal{X}$ and $\mathcal{Y}$, respectively. Given f and a loss function $L : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}_0^+$, we assess the fitting quality of the model on the data set $\mathcal{D}$ using the *empirical risk* $\mathcal{R}_\text{emp}(f) = n^{-1} \sum_{(\boldsymbol{x},y) \in \mathcal{D}} L(y, f(\boldsymbol{x}))$. We further denote the *test data* as $\mathcal{D}_\text{test}$ with $\mathcal{D} \cap \mathcal{D}_\text{test} = \emptyset$, which is an additional data set held back for performance evaluation. In case early stopping or parameter tuning is performed, we split $\mathcal{D}$ into a *training data* set $\mathcal{D}_\text{train} \subset \mathcal{D}$, which is used to train the model and a *validation data* set $\mathcal{D}_\text{val} \subset \mathcal{D}$, which is used for determining, e.g., the stopping iteration or the model's performance for validation. If none of these subroutines is used, we refer to $\mathcal{D}$ as training data (as in Algorithm 2).

2.2 Base Learner

A base learner $b_k : \mathcal{X} \to \mathbb{R}$ is used to model the contribution of one or multiple features to the estimated model $\hat{f}$. Possible base learners range from simple models like a linear model on one feature to complex models such as deep decision trees including many features. While the presented adaptions for CWB work also for multivariate base learners such as tensor product splines, we restrict ourselves to univariate functions $b_k(\boldsymbol{x}, \boldsymbol{\theta}) = g_k(x)^\mathsf{T}\boldsymbol{\theta}, \boldsymbol{\theta} \in \mathbb{R}^{d_k}$, for demonstration purposes. The function $g_k : \mathbb{R} \to \mathbb{R}^{d_k}$ is a generic representation for modeling alternatives such as linear effects, categorical effects or smooth effects. For smooth effects, numerical features are transformed using a basis representation $g_k(x) = (B_{k,1}(x), \ldots, B_{k,d_k}(x))^\mathsf{T}$ with d_k basis functions, e.g., via univariate penalized B-splines (P-Splines; Eilers and Marx, 1996). This generic representation defines a *design matrix* $\boldsymbol{Z}_k = (g_k(x_j^{(1)}), \ldots, g_k(x_j^{(n)}))^\mathsf{T} \in \mathbb{R}^{n \times d_k}$ given by a feature vector $\boldsymbol{x}_j \in \mathbb{R}^n$. Note that the base learner b_k implicitly selects the feature(s) on which it operates without explicitly denoting the feature(s). In this paper with univariate base learners, exactly one feature is passed to the generic representation g_k. An important property induced by the choice of a linear base learner is that two base learners of the same type $b_k(\boldsymbol{x}, \boldsymbol{\theta})$ and $b_k(\boldsymbol{x}, \boldsymbol{\theta}^*)$ sum up to one base learner of the same type:

$$b_k(\boldsymbol{x}, \boldsymbol{\theta}) + b_k(\boldsymbol{x}, \boldsymbol{\theta}^*) = b_k(\boldsymbol{x}, \boldsymbol{\theta} + \boldsymbol{\theta}^*). \tag{1}$$

During boosting, CWB selects its next component from a pre-defined set of base-learners $\mathcal{B} = \{b_k\}_{k\in\{1,\dots,K\}}$, where K often equals the number of features p.

2.3 Componentwise Boosting

CWB estimates $\hat{f}$ using an iterative steepest descent minimization in function space. $\hat{f}^{[m]}$ denotes the prediction model after m iterations. In each step, the *pseudo residuals* $r^{[m](i)} = -\frac{\partial L\left(y^{(i)}, f\left(\boldsymbol{x}^{(i)}\right)\right)}{\partial f(\boldsymbol{x}^{(i)})}\Big|_{f=\hat{f}^{[m-1]}}$, $i \in \{1, \dots, n\}$ indicate a functional gradient, evaluated at training data points, where changing the outputs of $\hat{f}^{[m]}$ reduces the overall loss of our current model, w.r.t. to the given loss L the most. CWB initializes $\hat{f}^{[m]}$ with a loss-optimal constant model – also called an offset or intercept. In the mth iteration, all base learners in $\mathcal{B}$ are fitted against $\boldsymbol{r}^{[m]}$ via L2-loss, and the best candidate is selected to additively update $f^{[m]}$, controlled by a *learning rate* ν. This procedure is repeated M times or until a convergence criterion is met (e.g., using early stopping as described in Section 3.2.3). The details of CWB are given in Algorithm 1 in Appendix A.1.

Due to property (1) of linear base learners and the additive model update of CWB, the estimated parameter of each base learner can be summed up after M iterations, and each aggregated parameter $\hat{\boldsymbol{\theta}}_k = \sum_{m=1}^M \sum_{k=1}^K \mathbb{1}_{\{k=k^{[m]}\}} \hat{\boldsymbol{\theta}}^{[m]}$ with its base representation g_k can be interpreted as a partial effect of the feature modeled by base learner b_k.

2.4 Computational Complexity of CWB

The computational complexity of CWB is directly related to the computational complexity of fitting each base learner. All linear base learners $b_1, \dots, b_K$ must solve a system of linear equations $\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{Z}_k\boldsymbol{\theta}_k = \boldsymbol{Z}_k^\mathsf{T}\boldsymbol{r}^{[m]}$. For simplicity, assume that all base learners have the same number of parameters, i.e., $d_k = d$. Such systems are usually Cholesky decomposed with $\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{Z}_k = \boldsymbol{L}_k\boldsymbol{L}_k^\mathsf{T}$, and complexity $\mathcal{O}(d^3)$. Taking into account additional matrix operations $\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{Z}_k$ with d^2n, $\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{r}^{[m]}$ with dn, and forward/backward solving with $(d^2-d)/2$ operations, this yields a complexity of $\mathcal{O}(d^2n+d^3)$. When applied to all K base learners in M iterations, this yields $\mathcal{O}(MK(d^2n + d^3))$ (neglecting operations such as the calculations of pseudo residuals $\boldsymbol{r}^{[m]}$ and sum of squared errors SSE_k, or finding the best base learner $k^{[m]}$).

The above can be accelerated considerably by pre-calculating the (expensive) Cholesky decomposition once for every base learner before boosting. This reduces the computational complexity of CWB to $\mathcal{O}(K(d^2n + d^3))$. When taking also the forward/backward solving and calculation of $\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{r}^{[m]}$ as operations per iteration into account, the complexity when caching heavy operations reduces to $\mathcal{O}(K(d^2n + d^3) + MK(d^2 + dn))$.

3 A more Efficient Componentwise Boosting

3.1 Binning

In order to make CWB feasible for large data sets and reduce computational resources in general, we propose to combine CWB with binning. While the primary goal of binning is to reduce the memory consumption by representing numerical feature via discretization, this method can also accelerate the model fit.

3.1.1 Discretizing numerical Features

Binning discretizes a numerical feature into a smaller number of design point values. Usually, binned values are constructed as an equally spaced grid Lang et al. (2014) $z_j^{(i)} = \mathsf{min}(\boldsymbol{x}_j) + (i-1)/(n^*-1)(\mathsf{max}(\boldsymbol{x}_j) - \mathsf{min}(\boldsymbol{x}_j))$, $i = 1, \dots, n^*$ and then replace each value $x_j^{(i)}$ with its closest design point $z_j^{(i)}$. The number of design points n^* can be chosen arbitrarily. Wood et al. (2017) argue that the trade-off between data size and statistical error due to imprecise feature values is most adequate for $n^* = \sqrt{n}$.

Design points and index vector. Instead of storing the discretized feature vector

$\tilde{\boldsymbol{x}}_j = (\tilde{x}_j^{(1)}, \dots, \tilde{x}_j^{(n)})^\mathsf{T} \in \mathbb{R}^n$, it is sufficient to store the n^* values of $\boldsymbol{z}_j$ as well as an additional index vector $\boldsymbol{ind}_k = (ind_k^{(1)}, \dots, ind_k^{(n)})^\mathsf{T}$ (i.e., the assignment of each discretized feature value to its bin $\tilde{x}_k^{(i)} = z_j^{(ind_k^{(i)})}$). The index vector is calculated by setting $ind_k^{(i)} = 1$ if $x_j^{(i)} \in [z_j^{(0)}; z_j^{(1)} + m_2]$ and $ind_k^{(i)} = l$ if $x_j^{(i)} \in (z_j^{(l-1)} - m_1; z_j^{(l)} + m_2]$, where $m_1 = (z_j^{(l)} - z_j^{(l-1)})/2$ and $m_2 = (z_j^{(l+1)} - z_j^{(l)})/2$ is half the distance of a design point to its left/right neighbor. Hence, binning can be seen as a hash map where the index vector $\boldsymbol{ind}_k$ is the hash function, the design points $\boldsymbol{z}_j$ are the keys, and $\boldsymbol{x}_j$ are the entries.

Binning reduces the amount of required storage for variables. Instead of storing the $n \times d_k$ matrix $\boldsymbol{Z}_k$, a reduced $n^* \times d_k$ matrix $\tilde{\boldsymbol{Z}}_k^b = (g_k(z_j^{(1)}), \dots, g_k(z_j^{(n^*)}))^\mathsf{T}$ based on bins $\boldsymbol{z}_j$ is stored. $\boldsymbol{k}_k$ is used to assign the ith row $\tilde{\boldsymbol{Z}}_k^b(i) = g_k(z_j^{(i)})^\mathsf{T}$, $i = 1, \dots, n^*$ to the ith row $\tilde{\boldsymbol{Z}}_k(i) = g_k(\tilde{x}_k^{(i)})^\mathsf{T}$, $i = 1, \dots, n$ of the discretized feature $\tilde{\boldsymbol{x}}_j$ by $\tilde{\boldsymbol{Z}}_k(i) = \tilde{\boldsymbol{Z}}_k^b(ind_k^{(i)})$. Note that the same lookup can also be applied to categorical features without binning.

An analogy to binning are sparse data matrices, a widely used data representation. Similar to binning, sparse data matrices choose another representation by storing the row and column index of only the non-zero elements and the corresponding values (see, e.g., Duff et al., 1989). Using sparse matrices has two major advantages. First, this approach incurs reduced memory load, and second, optimized algorithms can be used to calculate matrix operations much faster. A specific example where sparse matrices are used in the context of CWB is to store the base representation $\boldsymbol{Z}_k$ of P-spline base learners, since $\boldsymbol{Z}_k$ contains mainly zeros. The fitting process is also accelerated, as $\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{r}^{[m]}$ is calculated for just the non-zero elements.

3.1.2 Matrix multiplications on binned Features

When fitting a penalized base learner in CWB, the two matrix operations $\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{W}_k\boldsymbol{Z}_k + \lambda\boldsymbol{D}_k$, with a symmetric penalty matrix $\boldsymbol{D}_k \in \mathbb{R}^{d_k \times d_k}$, and $\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{W}_k\boldsymbol{r}^{[m]}$ are required. We assume the weight matrix to be diagonal $\boldsymbol{W}_k = \mathsf{diag}(\boldsymbol{w}_k)$ with elements $\boldsymbol{w}_k = (w_k^{(i)}, \dots, w_k^{(n)})^\mathsf{T}$ and set $\boldsymbol{w}_k$ to a vector of ones if no weights are used, i.e., $\boldsymbol{W}_k \equiv \boldsymbol{I}_n$. While $\lambda\boldsymbol{D}_k$ is not affected by binning, $\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{W}_k\boldsymbol{Z}_k$ and $\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{W}_k\boldsymbol{r}^{[m]}$ can be computed more efficiently on the binned design matrix. Algorithm 1 describes corresponding matrix operations using the index vector $\boldsymbol{k}_k$

and reduced design matrix $\tilde{\boldsymbol{Z}}_k^b$.

Algorithm 1 Calculation of $\tilde{\boldsymbol{Z}}_k^\mathsf{T}\boldsymbol{W}_k\tilde{\boldsymbol{Z}}_k$ and $\tilde{\boldsymbol{Z}}_k^\mathsf{T}\boldsymbol{W}_k\boldsymbol{r}^{[m]}$ on binned design matrix $\tilde{\boldsymbol{Z}}_k^b$ and weight matrix $\boldsymbol{W}_k = \mathsf{diag}(\boldsymbol{w}_k)$.

Input: Design matrix $\tilde{\boldsymbol{Z}}_k^b$, weight vector $\boldsymbol{w}_k$, pseuro residuals $\boldsymbol{r}^{[m]}$, and index vector $\boldsymbol{ind}_k$
Output: $\tilde{\boldsymbol{Z}}_k^\mathsf{T}\boldsymbol{W}_k\tilde{\boldsymbol{Z}}_k$ and $\tilde{\boldsymbol{Z}}_k^\mathsf{T}\boldsymbol{W}_k\boldsymbol{r}^{[m]}$

```
1: procedure binMatMat($\tilde{\boldsymbol{Z}}_k^b, \boldsymbol{W}_k, \boldsymbol{ind}_k, \boldsymbol{w}_k$)
2:     Initialize with zero matrix $\boldsymbol{U} = \boldsymbol{0}_{d_k \times n^*}$
3:     for $i \in \{1, \dots, n\}$ do
4:         $\boldsymbol{U}(ind_k^{(i)}) \mathrel{+}= w_{i,j}\tilde{\boldsymbol{Z}}_k^b(k_k^{(i)})$
5:     return $\boldsymbol{U}\tilde{\boldsymbol{Z}}_k^b$
6: end procedure
```

```
1: procedure binMatVec($\tilde{\boldsymbol{Z}}_k^b, \boldsymbol{r}^{[m]}, \boldsymbol{ind}_k, \boldsymbol{w}_k$)
2:     Initialize with zero vector $\boldsymbol{u} = \boldsymbol{0}_{n^*}$
3:     for $i \in \{1, \dots, n\}$ do
4:         $u^{(ind_k^{(i)})} \mathrel{+}= w_k^{(i)} r^{[m](i)}$
5:     return $\tilde{\boldsymbol{Z}}_k^{b\mathsf{T}}\boldsymbol{u}$
6: end procedure
```

The matrix $\boldsymbol{U}$ and vector $\boldsymbol{u}$ act as intermediate results and are used for the final matrix-matrix and matrix-vector operation on the reduced design matrix $\tilde{\boldsymbol{Z}}^b$. Due to the discretization, the number of matrix product operations reduces from $\mathcal{O}(nd^2)$ to $\mathcal{O}(n^*d^2+n)$ when using a diagonal weight matrix (Li and Wood, 2020).

CWB applies binning on a base learner level. Therefore, each base learner that uses binning individually calculates the bin values $\boldsymbol{z}_j$, the reduced design matrix $\tilde{\boldsymbol{Z}}_k^b$, and the index vector $\boldsymbol{ind}_k$. The binMatMat in Algorithm 1 is first used in CWB when calculating the Cholesky decomposition $\boldsymbol{L}$ of $\tilde{\boldsymbol{Z}}_k^\mathsf{T}\boldsymbol{W}_k\tilde{\boldsymbol{Z}}_k$ and then caches results for later usage. To calculate $\hat{\boldsymbol{\theta}}^{[m]}$ in each iteration, binMatVec in Algorithm 1 is used.

3.1.3 Computational Complexity when applied in CWB

Using Algorithm 1, the number of operations for calculating $\tilde{\boldsymbol{Z}}_k^\mathsf{T}\boldsymbol{W}_k\tilde{\boldsymbol{Z}}_k$ reduces from $K(d^2n+d^3)$ to $K(n^*d^2+n+d^2)$. In comparison to a routine without binning, this is a reduction in operations if $n^* < n(d^2-1)/d^2$ and $d > 1$. A similar result holds during the fitting process when applying binMatVec of Algorithm 1. Here, binning requires $n + dn^*$ operations, whereas a calculation with dense matrices requires dn operations. This is a reduction in operations if $n^* < n(d-1)/d$ and $d > 1$. Applying this for all K base learners in each of the M iterations yields a complexity of $\mathcal{O}(MK(d^2+dn^*+n)$. All in all, we obtain a computational complexity of $\mathcal{O}(K(d^2n^*+n+d^3)+MK(d^2+dn^*+n))$ instead of $\mathcal{O}(K(d^2n+d^3)+MK(d^2+dn))$. For the important case of P-spline base learners with

$d = 20$ as a typical choice of basis dimension and $n^* = \sqrt{n}$, the conditions for a reduction in operations are fulfilled (see also Section 4 for the effect of binning in practice).

3.2 Accelerating the Fitting Process of CWB

In risk minimization, standard gradient descent can evolve slowly if the problem is ill-conditioned (Ruder, 2016). To reduce this problem, momentum adds a fraction of the previous gradient to the update step for an accelerated fitting procedure (Qian, 1999). A further extension is Nesterov accelerated gradient (NAG; Nesterov, 1983), also known as Nesterov momentum. NAG performs a look ahead on what the gradient descent step is doing and adjusts the update step to improve convergence. To incorporate NAG into CWB, we follow Lu et al. (2020) and introduce a second base learner $b_{k_{\text{cor}}^{[m]}}$ that is fitted to the so-called *error-corrected pseudo residuals* $c^{[m](i)} = r^{[m](i)} + \frac{m}{m+1}(c^{[m-1](i)} - b_{k_{\text{cor}}^{[m-1]}}(\boldsymbol{x}^{(i)}, \hat{\boldsymbol{\theta}}_{\text{cor}}^{[m]}),\ i \in \{1, \dots, n\}$. Due to the recursive definition, the error-corrected pseudo residual $c^{[m](i)}$ at iteration m contains information of all previous pseudo residuals. The sequence $\{b_{k_{\text{cor}}^{[1]}}, \dots, b_{k_{\text{cor}}^{[m]}}\}$ of base learners then accumulates to the *momentum model* $h^{[m]} = h^{[m-1]} + \eta_m b_{k_{\text{cor}}^{[m]}}$, where $\eta_m = \beta\gamma\vartheta_m^{-1}$ is the learning rate of the momentum model. This learning rate contains the momentum parameter $\gamma \in \mathbb{R}_+$ and a sequence $\vartheta_m = 2/(m+1)$, which is later used to combine the *primary model* f and momentum model h. In contrast to standard CWB, the pseudo residuals $r^{[m](i)} = -\frac{\partial L(y^{(i)}, f(\boldsymbol{x}^{(i)}))}{\partial g(\boldsymbol{x}^{(i)})}\Big|_{g=g^{[m-1]}},\ i \in \{1, \dots, n\}$ are calculated as the gradient w.r.t. a convex combination $g^{[m]} = (1 - \vartheta_m)f^{[m]} + \vartheta_m h^{[m]}$ of the primary model $f^{[m]}$ and the momentum model $h^{[m]}$. The primary model $f^{[m]} = g^{[m-1]} + \beta b_{k^{[m]}}$ is calculated by adding the new component $b_{k^{[m]}}$ to the combined model $g^{[m]}$. Algorithm 2 summarizes the *accelerated CWB* (ACWB) algorithm. For simplicity, the loop to select the optimal base learner (lines 5 – 8 of Algorithm 1 Appendix A.1) is summarized as procedure $\mathsf{findBestBaselearner}(\boldsymbol{r}, \mathcal{B})$, which returns a tuple $(\hat{\boldsymbol{\theta}}^{[m]}, k^{[m]})$ of the estimated parameters $\hat{\boldsymbol{\theta}}^{[m]}$ and the index $k^{[m]}$ of the best base learner from set $\mathcal{B}$ of base learners.

Algorithm 2 ACWB algorithm with input and output.

Input Data $\mathcal{D}$, learning rate ν, momentum parameter γ, number of boosting iterations M, loss function L, set of base learner $\mathcal{B}$
Output Prediction model $\hat{f}^{[M]}$ defined by fitted $\hat{\boldsymbol{\theta}}^{[1]}, \dots, \hat{\boldsymbol{\theta}}^{[M]}$ and $\hat{\boldsymbol{\theta}}_{\text{cor}}^{[1]}, \dots, \hat{\boldsymbol{\theta}}_{\text{cor}}^{[M]}$

```
1: procedure ACWB($\mathcal{D}, \nu, \gamma, L, \mathcal{B}$)
2:     Initialize $\hat{f}^{[0]}(\boldsymbol{x}) = \operatorname*{arg\,min}_{c\in\mathbb{R}} \mathcal{R}_{\text{emp}}(c, \mathcal{D})$;  $\hat{h}^{[0]}(\boldsymbol{x}) = \hat{f}^{[0]}(\boldsymbol{x})$;  $\hat{g}^{[0]}(\boldsymbol{x}) = \hat{f}^{[0]}(\boldsymbol{x})$;
3:     for $m \in \{1, \dots, M\}$ do
4:         $\vartheta_m = \frac{2}{m+1}$
5:         $\hat{g}^{[m]}(\boldsymbol{x}) = (1 - \vartheta_m)\hat{f}^{[m-1]}(\boldsymbol{x}) + \vartheta_m \hat{h}^{[m-1]}(\boldsymbol{x})$
6:         $r^{[m](i)} = - \left.\frac{\partial L(y^{(i)}, f(\boldsymbol{x}^{(i)}))}{\partial g(\boldsymbol{x}^{(i)})}\right|_{g=\hat{g}^{[m]}}, \quad \forall i \in \{1, \dots, n\}$
7:         $(\hat{\boldsymbol{\theta}}^{[m]}, k^{[m]}) = \mathsf{findBestBaselearner}(\boldsymbol{r}^{[m]}, \mathcal{B})$
8:         $\hat{f}^{[m]}(\boldsymbol{x}) = \hat{g}^{[m]}(\boldsymbol{x}) + \nu b_{k^{[m]}}(\boldsymbol{x}, \hat{\boldsymbol{\theta}}^{[m]})$
9:         if $m > 1$ then
10:            $c^{[m](i)} = r^{[m](i)} + \frac{m}{m+1}\left(c^{[m-1](i)} - b_{k_{\text{cor}}^{[m-1]}}(\boldsymbol{x}, \hat{\boldsymbol{\theta}}_{\text{cor}}^{[m]}\right), \quad \forall i \in \{1, \dots, n\}$
11:        else
12:            $\boldsymbol{c}^{[m]} = \boldsymbol{r}^{[m]}$
13:        end if
14:        $(\hat{\boldsymbol{\theta}}_{\text{cor}}^{[m]}, k_{\text{cor}}^{[m]}) = \mathsf{findBestBaselearner}(\boldsymbol{c}^{[m]}, \mathcal{B})$
15:        $\eta_m = \gamma\nu\vartheta_m^{-1}$
16:        $\hat{h}^{[m]}(\boldsymbol{x}) = \hat{h}^{[m-1]}(\boldsymbol{x}) + \eta_m b_{k_{\text{cor}}^{[m]}}(\boldsymbol{x}, \hat{\boldsymbol{\theta}}_{\text{cor}}^{[m]})$
17:    end for
18:    return $\hat{f} = \hat{f}^{[M]}$
19: end procedure
```

3.2.1 Retaining CWB Properties

ACWB fits a second base learner $b_{k_{\text{cor}}^{[m]}}$ in order to accelerate the fitting process with the momentum model h. In addition to the *fitting trace* for the primary model $\Theta_f = \{\hat{\boldsymbol{\theta}}^{[1]}, \dots, \hat{\boldsymbol{\theta}}^{[M]}\}$, another fitting trace $\Theta_h = \{\hat{\boldsymbol{\theta}}_{\text{cor}}^{[1]}, \dots, \hat{\boldsymbol{\theta}}_{\text{cor}}^{[M]}\}$ is also stored for the momentum model. Consequently, ACWB contains twice as many base learners as CWB after m iterations. Despite the more complex fitting routine, the parameters for ACWB can still be additively updated. Suppressing the iteration m for readability, we assign $(1 - \vartheta_m)\hat{\boldsymbol{\theta}}_f + \vartheta_m \hat{\boldsymbol{\theta}}_h$ as the current $\hat{\boldsymbol{\theta}}_g$, update $\hat{\boldsymbol{\theta}}_f$ to $\hat{\boldsymbol{\theta}}_g + \beta(0 \cdots \hat{\boldsymbol{\theta}}^{[m]\mathsf{T}} \cdots 0)^{\mathsf{T}}$, and update $\hat{\boldsymbol{\theta}}_h$ to $\hat{\boldsymbol{\theta}}_h + \eta_m(0 \cdots \hat{\boldsymbol{\theta}}_{\text{cor}}^{[m]\mathsf{T}} \cdots 0)^{\mathsf{T}}$. The initial parameters $\hat{\boldsymbol{\theta}}_f$ and $\hat{\boldsymbol{\theta}}_h$ for $m = 0$ are set to zero. An additive aggregation of parameters is important to retain interpretable additive partial effects and allows for much faster predictions.

3.2.2 Computational Complexity of ACWB

As derived in Section 2.4, the complexity of CWB is $\mathcal{O}(K(d^2n + d^3) + MK(d^2 + dn))$ and hence scales linearly in the number of rows n and number of base learners K. By fitting a second base learner in each iteration, the complexity for ACWB during the fitting process is doubled compared to CWB, while costs for expensive pre-calculation steps do not change. This results in a complexity of $\mathcal{O}(K(d^2n + d^3) + 2MK(d^2 + dn))$.

3.2.3 A Hybrid CWB Approach

Algorithm 3 HCWB as a combination of ACWB and CWB. $\mathcal{R}_\text{emp}(f|\mathcal{D}_\text{val})$ denotes the empirical risk calculated on validation data $\mathcal{D}_\text{val}$.

Input Data $\mathcal{D} = \mathcal{D}_\text{train} \cup \mathcal{D}_\text{val}$, learning rate ν, momentum parameter γ, number of boosting iterations M, patience parameter pat_0, loss function L, set of base learner $\mathcal{B}$

Output Prediction model $\hat{f}^{[M]}$ defined by parameters $\hat{\boldsymbol{\theta}}^{[1]}, \dots, \hat{\boldsymbol{\theta}}^{[M]}$ and $\hat{\boldsymbol{\theta}}_\text{cor}^{[1]}, \dots, \hat{\boldsymbol{\theta}}_\text{cor}^{[M]}$

```
1:  procedure HCWB($\mathcal{D}, \nu, \gamma, pat_0, L, \mathcal{B}$)
2:      $m = 0$
3:      for $m = 1, \dots, M$ do
4:          if $pat_0 < pat$ then
5:              $\hat{f}^{[m]} = \mathsf{updateACWB}(\hat{f}^{[m-1]}, \mathcal{D}_\text{train}, \nu, \gamma, L, \mathcal{B})$
6:              if $\mathcal{R}_\text{emp}(\hat{f}^{[m]}|\mathcal{D}_\text{val}) > \mathcal{R}_\text{emp}(\hat{f}^{[m-1]}|\mathcal{D}_\text{val})$ then $pat$ += 1 else $pat = 0$
7:          else
8:              $\hat{f}^{[m]} = \mathsf{updateCWB}(\hat{f}^{[m-1]}, \mathcal{D}_\text{train}, \nu, L, \mathcal{B})$
9:          end if
10:     end for
11:     return $\hat{f} = \hat{f}^{[m]}$
12: end procedure
```

It is well known that excessive training boosting can lead to overfitting (see, e.g., Grove and Schuurmans, 1998; Jiang et al., 2004). This can be controlled by stopping the algorithm early, which works irrespective of using gradient descent with or without NAG. However, as shown recently by Wang et al. (2020, Theorem 4), gradient methods with NAG might diverge for noisy convex problems. Whereas inexact gradients induced by noise terms influence the convergence rates only by an additional constant when using no acceleration, NAG potentially diverges with a rate that increases with the number of iterations. To over-

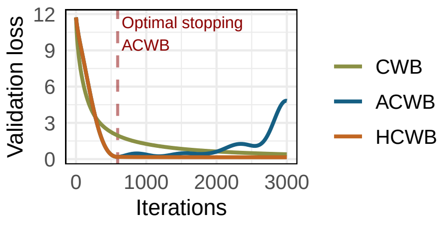

Figure 1: Exemplary course of the empirical risk for different CWB variations on a test set. The vertical dashed line indicates the optimal iteration found by ACWB. All methods are trained for 3000 iterations.

come this issue, we propose a hybrid approach by starting the fitting process with ACWB on data $\mathcal{D}_{\text{train}} \subset \mathcal{D}$ until it is stopped early using a validation set $\mathcal{D}_{\text{val}}$ ($\mathcal{D}_{\text{val}} \cap \mathcal{D}_{\text{train}} = \emptyset$ and $\mathcal{D}_{\text{val}} \cup \mathcal{D}_{\text{train}} = \mathcal{D}$). Thereafter, the training is continued to fine-tune the predictions using the classical CWB on all available data $\mathcal{D}$ until M iterations are reached (or, alternatively, using again a train-validation split and early stopping). Algorithm 3 describes this hybrid approach. The update routines `updateCWB` and `updateACWB` describe the fitting component of the respective algorithms (CWB; Algorithm 1 Appendix A.1 lines 4 – 10, ACWB; Algorithm 2 lines 4 – 16). The example shown in Figure 1 depicts the behavior of our proposed adaption. After early stopping ACWB, we can continue to further decrease the validation error, whereas continuing training with acceleration (ACWB) increases the validation error. HCWB thus improves the performance of ACWB while requiring less iterations compared to CWB with a similar validation error. In Section 4.1, we investigate computational advantages of the hybrid approach in terms of runtime, memory consumption, and estimation properties. Appendix B.3 further demonstrates how the reduction in space complexity of binning allows users to fit CWB even on very large data sets with reasonable computing resources (e.g., with millions of observations and a data size of ≈ 3 GB using only 32 GB of RAM). Without binning, the model allocates too much memory during its initialization and subsequently crashes.

4 Experiments

We study the efficacy of the proposed improvements on simulated and real-world data (cf. Appendix B.5). All CWB variants (CWB, ACWB, and HCWB) are either denoted with *(nb)* if no binning is applied, or *(b)* if binning is used. The hyperparameters (HPs) of our proposed algorithm are the amount of binning, the momentum rate γ, and the patience parameter pat_0. Further HPs for all CWB variants (including vanilla CWB) are the learning rate ν, the degrees of freedom df, and the number of boosting iterations M. As simulation studies and real-world applications have different purposes, we will define separate HP tuning schemes for both but set the number of bins to $n^* = \sqrt{n}$, as suggested by Wood et al. (2017), and the patience parameter to $pat_0 = 5$ for all experiments. HPs defining the flexibility of base learners such as number of bases or the degree of basis functions for P-Spline base learners are set to 20 base functions and degree 3, respectively.

In Section 4.1, we first use simulated data to investigate the following hypotheses:

H1 **(memory and runtime efficiency of binning):** Compared to CWB (nb), binning reduces memory consumption and runtime.

H2 **(partial effects of binning):** Compared to CWB (nb), using CWB (b) does not deteriorate the estimation quality of partial effects.

H3 **(iterations and partial effects of accelerated methods):** Compared to CWB (nb), ACWB (nb) and HCWB (nb) require fewer iterations to achieve almost identical partial effect estimates.

H4 **(scaling behaviour of ACWB/HCWB):** Empirical runtimes of binning and ACWB follow the previously derived complexity conditions and thus scale efficiently with an increasing number of observations and base learners.

In addition to the simulation study, we conduct a comparison of CWB and HCWB on real-world data sets in Section 4.2. Here, we investigate the following experimental questions:

EQ1 **(implementation comparison with the state-of-the-art software `mboost`):** Compared to `mboost`, our implementation `compboost` is less time-consuming.

EQ2 (algorithmic comparison of accelerated methods): When compared to vanilla CWB (nb), the prediction performance of the five proposed CWB variants (CWB (b), ACWB/HCWB (b)/(nb)) does not deteriorate while yielding faster runtimes.

EQ3 (comparison with state-of-the-art algorithms) The prediction performance of ACWB (b) and HCWB (b) is competitive with state-of-the-art boosting algorithms while yielding notably faster runtimes.

All simulations and benchmarks are conducted using `R` (R Core Team, 2021) version 3.6.3 on three identical servers with 32 2.60 GHz Intel(R) Xeon(R) CPU e5-2650 v2 processors. Reproducibility is ensured by providing a Docker image with pre-installed software and benchmark code. References to GitHub containing our software `compboost` and the benchmark source code with results are referenced in the Supplementary Material.

4.1 Numerical Experiments on Simulated Data

4.1.1 Simulation Setup

We define the number of informative and noise features as p and p_{noise}, respectively. Numerical features $\boldsymbol{x}_j$ are simulated by drawing the minimum value $x_{j,\text{min}}$ from a uniform distribution $U[0, 100]$ and the maximal value $x_{j,\text{max}} = x_{j,\text{min}} + \rho_k$, where ρ_k also follows a uniform distribution $U[0, 100]$. The n feature values $\boldsymbol{x}_j$ are simulated i.i.d. from $U[x_{j,\text{min}}, x_{j,\text{max}}]$. All feature effects are simulated as non-linear effects using splines. The spline basis $\boldsymbol{Z}_k$ for the simulation is created using feature $\boldsymbol{x}_j$ with splines of order 4 and 10 base functions. To obtain unique splines for each $\boldsymbol{x}_j$, we sample $\tau_k \sim \mathcal{N}_{10}(0, 9)$ and define the jth feature effect as $\boldsymbol{\eta}_k = \boldsymbol{Z}_k \boldsymbol{\tau}_k$. Each of the p_{noise} noise features $\dot{\boldsymbol{x}}_j$ are drawn from a standard normal distribution. The final data set is given as $\mathcal{D} = \{(x_1^{(i)}, \dots, x_p^{(i)}, \dot{x}_1^{(1)}, \dots, \dot{x}_{p_{\text{noise}}}^{(i)}, y^{(i)}) \mid i = 1, \dots, n\}$ with target vector $\boldsymbol{y} \sim \mathcal{N}_n(\boldsymbol{\eta}, \mathsf{diag}_n\,(\hat{\sigma}^2(\boldsymbol{\eta})/\text{SNR}^2))$, where SNR is the *signal-to-noise* ratio, $\boldsymbol{\eta} = \sum_{j=1}^{p} \boldsymbol{\eta}_k$ and $\hat{\sigma}(\boldsymbol{\eta})$ the sample variance of $\boldsymbol{\eta}$. Different values of SNR can be used to test CWB on regression tasks of varying hardness.

The experimental design is defined on a grid with $n \in \{5000, 10000, 20000, 50000, 100000\}$, $p \in \{5, 10, 20, 50\}$, $p_{\text{noise,rel}} \in \{0.5, 1, 2, 5\}$ with $p_{\text{noise}} = p \cdot p_{\text{noise,rel}}$, and SNR $\in \{0.1, 1, 10\}$.

The choices of n, p, and p_{noise} are particularly relevant for memory and runtime investigations (H1) and (H4). For each combination of experimental settings, we draw 20 different target vectors for statistical replications of each scenario. When measuring the memory consumption, we only use one statistical replication of each configuration, as the memory consumption is almost identical for all repetitions of one configuration. We use `valgrind` (Nethercote and Seward, 2007) to measure the allocated memory. For memory and runtime comparisons and thus also for complexity considerations, the number of boosting iterations is set to 200. In order to assess the estimation performance of partial effects (H2) and (H3), we use the *mean integrated squared error* $\text{MISE} = \frac{1}{p}\sum_{j=1}^{p}\int_{\mathsf{min}(\boldsymbol{x}_j)}^{\mathsf{max}(\boldsymbol{x}_j)}(b_k(x,\boldsymbol{\theta}_k) - b_k(x,\hat{\boldsymbol{\theta}}_k))^2dx$ between the estimated base learner $b_k(\boldsymbol{x},\hat{\boldsymbol{\theta}}_k)$ and the true effect $b_k(\boldsymbol{x},\boldsymbol{\theta}_k)$ given by the randomly generated spline.

For performance comparisons, we use CWB with early stopping based on a large, noise-free data set. This ensures correct early stopping and allows us to draw conclusions on CWB's estimation performance without additional uncertainty induced by finding the optimal stopping iteration. We evaluate different momentum learning rates on a uniform grid from 10^{-1} to 10^{-7}. For CWB as well as for ACWB and HCWB, we set the learning rate to $\nu = 0.05$ as suggested in the literature (Bühlmann et al., 2007) and ensure unbiased feature selection (see Appendix A.2) by setting the degrees of freedom of each base learner to $\text{df} = 5$, which provides just enough flexibility for our simulated non-linear functions. The number of boosting iterations are fixed for H1 and dynamically found for H2 and H3.

4.1.2 Results

H1 (memory and runtime efficiency of binning). Figure 2 illustrates the faster runtime as well as the savings in memory consumption w.r.t. n and p. With binning, CWB is four times faster for smaller data sets and up to six times faster for larger data sets. For smaller data sets, binning improves the memory consumption only slightly. Improvements do become large enough to be meaningful when the model is trained on larger data sets and/or more features, consuming only up to a seventh of the original memory usage. Appendix B.7.1 contains an empirical verification of the computational complexity reported in Section 3.1.3 using the results from H1.

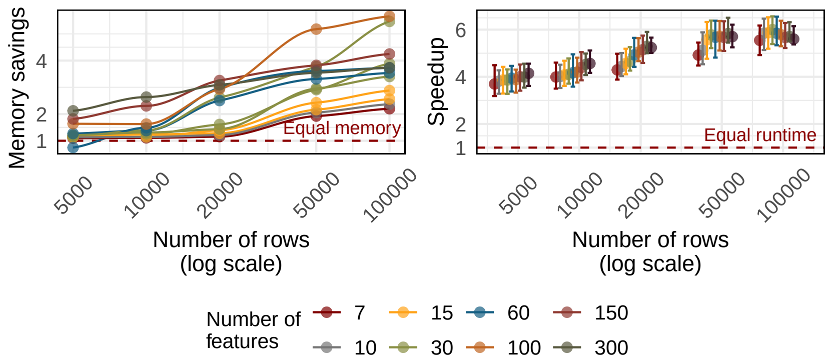

Figure 2: Relative memory savings (left) and speedup as relative runtime (right) when binning is applied compared to no binning (H1). Memory savings and speedup are defined as a ratio of memory or time used by the original algorithm without binning divided by the memory consumption or runtime when binning is applied. The dashed horizontal line indicates an equal memory consumption or runtime of both methods.

H2 (partial effects of binning). The left plot of Figure 3 shows one exemplary comparison of the true effect, the estimated effect using binning, and the estimated effect without binning. The MISE is visualized by the shaded area between the true and estimated function. As shown on the right in Figure 3, there is no notable difference between the MISE whether binning is applied or not. In order to also examine the efficacy of binning in particularly challenging data situations, Appendix B.1 investigates how (highly) skewed feature distributions and outliers affect the partial effects estimation performance. Experiments show that binning works equally well compared to no binning for skewed feature distributions, which can be attributed to the equidistant bin values that naturally preserve the underlying distribution. Results further suggest that binning can break down if outliers occur. But this effect is only observable in small sample size situations, whereas we mainly propose this technique for large amount of samples.

H3 (iterations and partial effects of accelerated methods). Figure 4 (left) shows the difference of MISE values of CWB compared to HCWB and ACWB. As hypothesized, partial effect estimation of ACWB is inferior to CWB due to its acceleration and potentially

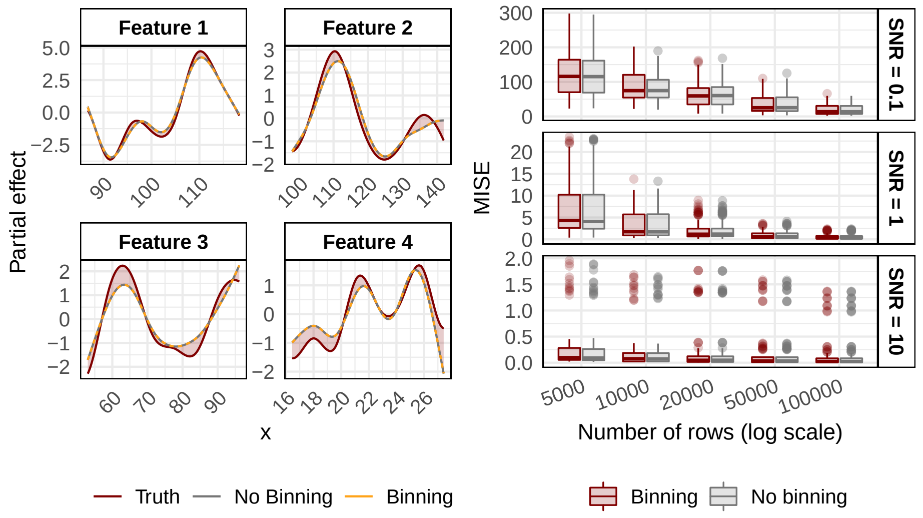

Figure 3: Left: Exemplary estimated partial and true effects for 100000 observations, 4 features, and a SNR of 0.1. The MISE of the curves are 49.08 for CWB and 49.10 for CWB with binning. Right: Comparison of the MISE when using binning vs. no binning.

stopping too early or overshooting the optimal solution. This difference is negligible for small momentum values, but not for higher values. It is worth noting that the SNR in this cases is rather large, potentially undermining the effect of residual-correcting base learners of ACWB. Based on the given results, we recommend a default momentum value of $\gamma = 0.0034$ for ACWB, yielding small MISE differences while simultaneously maximizing the speedup. In contrast, HCWB performs as strongly as CWB or even better for settings with more noise (smaller SNR). The right pane of Figure 4 shows the relative ratio of early stopping iterations between CBW and HCWB or ACWB. In almost all cases, the accelerated versions require fewer iterations than CWB. This is especially striking for higher momentum values, and hence we suggest a default of $\gamma = 0.037$ for HCWB. In settings with moderate or large SNR, HCWB requires as many iterations as CWB.

H4 (scaling behavior of ACWB/HCWB). Next, we empirically investigate runtimes to scrutinize our derivations in Section 3.1.3 and 3.2.2. To this end, we use the

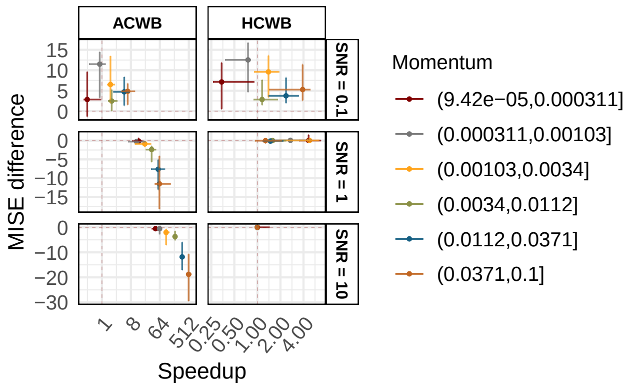

Figure 4: Left: MISE difference between CWB and ACWB/HCWB for partial effect estimation for different momentum values (colors) and SNR (rows). A negative difference indicates that CWB has a lower MISE and therefore a better estimation. Right: Multiplicative factor of iterations needed to train CWB compared to ACWB/HCWB, categorized by the momentum parameter (colors) and SNR (facets). A value of 10 indicates that ACWB/HCWB requires 10 times fewer iterations than CWB until convergence.

available runtimes, fit a model for each complexity statement to the given data, and compare the estimates to the theoretically derived factors in our complexity analyses. The full results are given in Appendix B.7. In summary, for both binning and ACWB, fitted models yield an (almost) perfect fit with an R^2 of 1 and 0.975, respectively, thus underpinning our complexity estimates with only smaller deviations from theoretical numbers. In particular, this confirms the presumed speedup when using binning and the efficiency of ACWB, scaling linearly in n and K.

4.2 Benchmark on Real-World Data

4.2.1 Setup

Algorithms: We compare our CWB variants with XGBoost (Chen et al., 2018) and EBM (Nori et al., 2019). XGBoost represents an efficient and well-performing state-of-the-art implementation of gradient boosting with tree base learners (Friedman and Hastie, 2001). EBM is used to compare against a recent and interpretable boosting method. Like

CWB, EBM is based on additive models with additional pairwise interactions, but instead uses a different fitting technique based on a round robin selection of base learners. Although the model can be interpreted by looking at partial effects, some other key features of CWB such as unbiased feature selection cannot be directly transferred to EBM.

Benchmark settings: For performance comparisons in EQ2 and EQ3, we use the area under the ROC curve (AUC) based on a 5-fold cross validation (CV). Additionally, we employ nested resampling to ensure unbiased performance estimation (Bischl et al., 2012) for EQ3. In the inner loop, models are tuned via Hyperband (Li et al., 2017) with the number of boosting iterations as budget parameter. We start at 39 iterations and double iterations until 5000 iterations are reached. This results in 314 different HP configurations for each learner. Each of these HP configuration is evaluated using a 3-fold CV (the inner CV loop). A table with all learners, corresponding software packages, and HP spaces from which each HP configuration is sampled is given in Appendix B.4.

Used software: All experiments are executed using `R` (R Core Team, 2021). The packages used for the benchmark are `mlr3` (Lang et al., 2019) as a machine learning framework with extensions, including `mlr3tuning` (Becker et al., 2021) for tuning, `mlr3pipelines` (Binder et al., 2021) for building pre-processing pipelines such as imputation or feature encoding, and `mlr3hyperband` (Becker et al., 2021). The package `interpret`, which implements EBM, was used by calling the Python implementation (Nori et al., 2019) using `reticulate` (Ushey et al., 2020) to run EBM with its full functionality. The HP space of XGBoost is defined as the "simple set" suggested in Thomas et al. (2018).

4.2.2 Results

EQ1 (implementation comparison with the state-of-the-art CWB implementation `mboost`). A comparison of our vanilla CWB implementation `compboost` with the state-of-the-art implementation `mboost` already reveals a speedup of 2 to 4 using `compboost` (by outsourcing various functionalities to `C++`). When additionally using acceleration methods and binning, an increase of the speedup up to a factor of 30 for MiniBooNE and Albert can be achieved. As CWB (nb) in `compboost` and `mboost` implement the same algorithm, they are equivalent in their predictive performance. Full details are given in Appendix B.9.

EQ2 (algorithmic comparison of accelerated methods). As shown in Figure 5, HCWB learns faster than CWB due to the acceleration and higher momentum. Performance improvements of ACWB take longer but surpass CWB on four out of the six data sets. As expected, the AUC of ACWB starts to decrease after the optimal number of boosting iterations is reached, while HCWB corrects this overly aggressive learning behavior by switching to CWB. The runtime of ACWB is about twice as high as for CWB due to the second error-correcting base learner fitted in each iteration (Algorithm 2 line 14). Traces for binning look similar to no binning but do exhibit shorter training times, which underpins the effectiveness of binning.

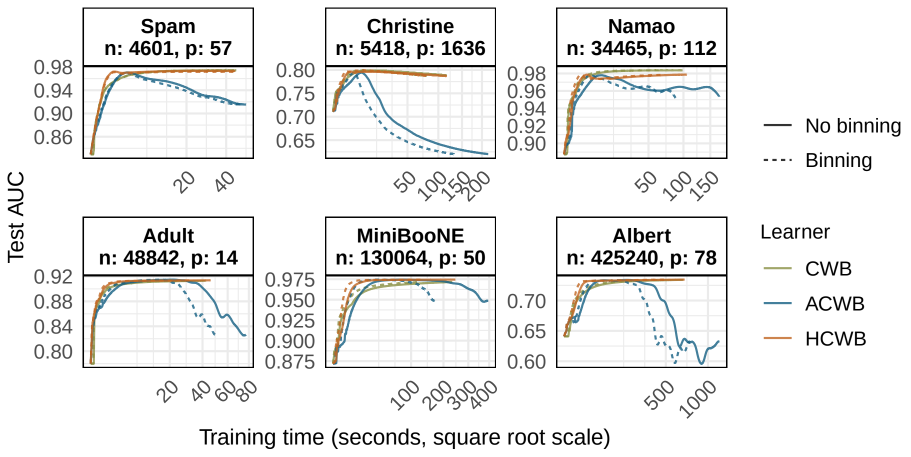

Figure 5: Test AUC traces of all CWB variants over 5000 boosting iterations without early stopping. Each trace is calculated as an average over the 5 runs in the 5-fold CV.

Figure 6 additionally shows test AUC and runtimes of all CWB variants when using early stopping based on a validation data set $\mathcal{D}_{\text{val}}$ (defined as 30% of $\mathcal{D}$). To check if performance changes between different models are significant, we use the resulting AUC values and compute a beta regression with learners as covariates and the AUC as a response variable (see Appendix B.11). Both ACWB (p-value = 0.4122) and HCWB (p-value = 0.6927) do not yield a significantly smaller AUC value. Furthermore, results show that binning does also not have a significant effect on the performance (p-value = 0.9594). At the same time, binning improves the runtime by an average speedup of 1.5 for all three CWB variants (cf. Appendix B.10 Table 4). ACWB and HCWB even yield further improvements with an average speedup of 3.8 and 2.38, respectively.

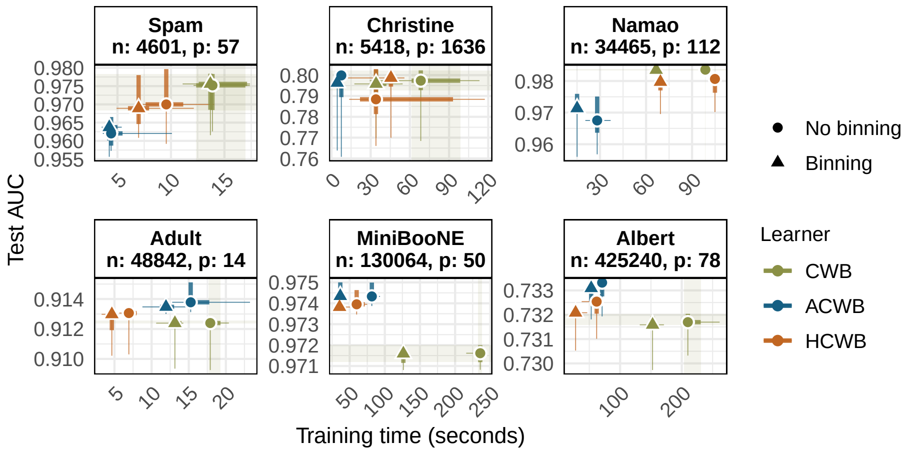

Figure 6: Scatter plot of average AUC values and training times for all CWB variants (color / symbol). Horizontal and vertical boxes indicate the 25- and 75%-quantile, and colored lines indicate the possible range of values. Shaded areas additionally highlight the 25- and 75%-quantiles of CWB (nb) to facilitate easy comparison of other CWB variants with this baseline.

EQ3 (comparison with state-of-the-art algorithms). Figure 7 shows that state-of-the-art algorithms benefit from their more complex model structure by also considering complex feature interactions (EBM allows for interactions by design, while XGBoost uses tree base learners, which induce more complex interactions with larger tree depth). The improvement in AUC of these methods compared to CWB is only practically relevant for the data set Christine, which shows an AUC increase of 3.42% for XGBoost. The AUC improvement for all other tasks is (notably) smaller than 3%, even though our approach uses a fully interpretable model. In terms of runtime, ACWB (b) and HCWB (b) outperform XGBoost and EBM on most data sets. On Spam and Christine, XGBoost is faster than our algorithms, which we attribute to their small sample size. In general, ACWB (b) and

HCWB (b) are 4.62 times faster than EBM and 1.66 times faster than XGBoost. The total runtimes are reported in Appendix B.12 Table 7.

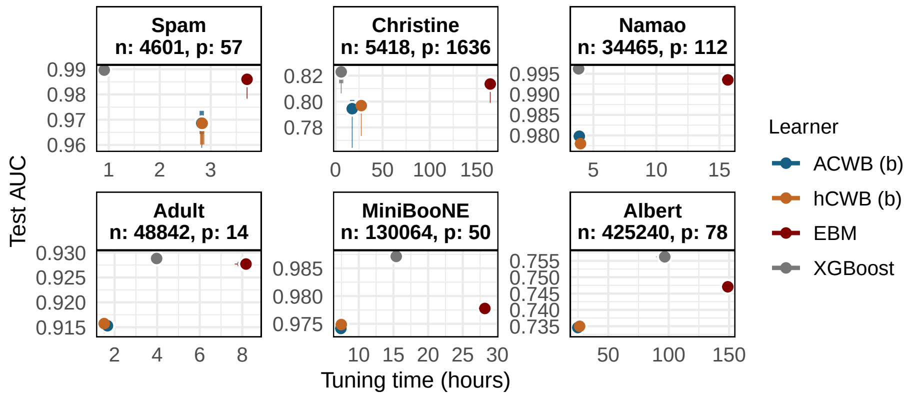

Figure 7: AUC and training time of ACWB (b), HCWB (b), XGBoost, and EBM.

The given results further demonstrate the speed up of our approach in a real-world application that includes tuning costs. E.g., for the Albert data set binned HCWB has a total tuning time of 5.5 days (26.3 hours on average times five outer folds). In contrast, CWB without binning would result in a total tuning time of 38.4 days (based on the reported speedup in EQ2). Hence, with our adaptions, we are able to reduce the total tuning time from over a month, which many users might deem as unacceptable, to under a week.

5 Conclusion

Adaptions to CWB presented in this paper can notably improve the use of computational resources, reducing runtimes by up to a factor of 6 and memory usage up to factor of 4. Incorporating binning with equally spaced design points efficiently leverages CWB's base learner structure with one feature per base learner. Benchmark results show that binning reduces the training time without impairing the predictive performance. The proposed accelerated CWB algorithm (ACWB) is furthermore a natural extension of CWB and provides a faster training procedure at the expense of a potential deterioration of predictive performance when not stopped properly. Our alternative hybrid solution (HCWB) accounts for this and presents no drawbacks in comparison to the standard CWB algorithm. However, HCWB does incur slightly longer runtimes compared to ACWB. In practice, HCBW yields good out-of-the-box performance, while ACWB does not require an additional validation data set and can thus be beneficial in low sample size regimes.

Practical recommendations for hyperparameters. In practice, CWB and the proposed variants in this work yield good performance out-of-the-box by choosing a learning rate $\nu \in [0.01, 0.1]$ (see, e.g., Bühlmann et al., 2007), a moderate number of bases functions per spline, e.g., 20, and equal degrees of freedom (e.g., df $= 8$, see Appendix A.2 for more details) for all splines. Our experiments further show that good momentum defaults are $\gamma = 0.0034$ for ACWB and $\gamma = 0.037$ for HCWB. The number of boosting iterations M can be found by early stopping. This leaves no or only M as tuning parameter and puts CWB on a similar level of practicability as other fitting routines for GAMs. While our provided defaults should work well in most cases, users that want to tune HPs are referred to our experimental section, where we provide meaningful ranges for all parameters, in particular the momentum, and show how much can be gained from tuning these values.

Future research will investigate how the proposed framework can be used and further improved for more complex additive models structures in both the predictors as well as in the outcome, e.g., for functional regression models.

ACKNOWLEDGMENT

This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A and Federal Ministry for Research and Technology (BMFT) under Grant FKZ: 01ZZ1804C (DIFUTURE, MII). The authors of this work take full responsibilities for its content.

DISCLOSURE STATEMENT

The authors report there are no competing interests to declare.

SUPPLEMENTARY MATERIAL

Appendix: Descriptions of possible categorical feature representations with a short comparison w.r.t. runtime and memory consumption as well as class selection properties in the presence of noise. The Appendix further contains empirical validation of the computational complexity estimates as given in Section 2.4 and 3.1.3. The appendix also contains a figure for the full benchmark.

Source code of `compboost`: `github.com/schalkdaniel/compboost` (Commit tag of the snapshot used in this paper: `c68e8fb32aea862750991260d243cdca1d3ebd0e`)

Benchmark source code: `https://github.com/schalkdaniel/cacb-paper-bmr`

Benchmark Docker: Docker image with pre-installed packages to run the benchmark and access results for manual inspection: `hub.docker.com/repository/docker/schalkdaniel/cacb-paper-bmr`

References

Becker, M., S. Gruber, J. Richter, J. Moosbauer, and B. Bischl (2021). *mlr3hyperband: Hyperband for 'mlr3'.* R package version 0.1.2.

Becker, M., M. Lang, J. Richter, B. Bischl, and D. Schalk (2021). *mlr3tuning: Tuning for 'mlr3'.* R package version 0.8.0.

Binder, M., F. Pfisterer, L. Schneider, B. Bischl, M. Lang, and S. Dandl (2021). *mlr3pipelines: Preprocessing Operators and Pipelines for 'mlr3'.* R package version 0.3.4.

Bischl, B., O. Mersmann, H. Trautmann, and C. Weihs (2012). Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary computation 20*(2), 249–275.

Brockhaus, S., D. Rügamer, and S. Greven (2020). Boosting functional regression models with fdboost. *Journal of Statistical Software 94*(10), 1–50.

Bühlmann, P., T. Hothorn, et al. (2007). Boosting algorithms: Regularization, prediction and model fitting. *Statistical science 22*(4), 477–505.

Bühlmann, P. and B. Yu (2003). Boosting with the L2 loss: regression and classification. *Journal of the American Statistical Association 98*(462), 324–339.

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

Chen, T., T. He, M. Benesty, V. Khotilovich, and Y. Tang (2018). *xgboost: Extreme Gradient Boosting.* R package version 0.6.4.1.

Duff, I. S., R. G. Grimes, and J. G. Lewis (1989). Sparse matrix test problems. *ACM Transactions on Mathematical Software (TOMS) 15*(1), 1–14.

Eilers, P. H. and B. D. Marx (1996). Flexible smoothing with B-splines and penalties. *Statistical science*, 89–102.

Freund, Y., R. E. Schapire, et al. (1996). Experiments with a new boosting algorithm. In *icml*, Volume 96, pp. 148–156. Citeseer.

Friedman, J. and T. Hastie (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 1189–1232.

Grove, A. J. and D. Schuurmans (1998). Boosting in the limit: Maximizing the margin of learned ensembles. In *AAAI/IAAI*, pp. 692–699.

Hofner, B., T. Hothorn, T. Kneib, and M. Schmid (2011). A framework for unbiased model selection based on boosting. *Journal of Computational and Graphical Statistics 20*(4), 956–971.

Hofner, B., A. Mayr, and M. Schmid (2016). gamboostLSS: An R package for model building and variable selection in the GAMLSS framework. *Journal of Statistical Software 74*(1).

Hothorn, T., P. Buehlmann, T. Kneib, M. Schmid, and B. Hofner (2020). mboost: Model-based boosting. R package version 2.9-2.

Jiang, W. et al. (2004). Process consistency for adaboost. *The Annals of Statistics 32*(1), 13–29.

Lang, M., M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl (2019). mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*.

Lang, S., N. Umlauf, P. Wechselberger, K. Harttgen, and T. Kneib (2014). Multilevel structured additive regression. *Statistics and Computing 24*(2), 223–238.

Li, L., K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research 18*(1), 6765–6816.

Li, Z. and S. N. Wood (2020). Faster model matrix crossproducts for large generalized linear models with discretized covariates. *Statistics and Computing 30*(1), 19–25.

Liew, B. X., D. Rugamer, A. Stocker, and A. M. De Nunzio (2020). Classifying neck pain status using scalar and functional biomechanical variables – Development of a method using functional data boosting. *Gait & posture 76*, 146–150.

Lu, H., S. P. Karimireddy, N. Ponomareva, and V. Mirrokni (2020). Accelerating Gradient Boosting Machines. In *International Conference on Artificial Intelligence and Statistics*, pp. 516–526.

Molnar, C. (2020). *Interpretable Machine Learning.* Lulu.com.

Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate $\mathsf{O}(1/k^2)$.

Nethercote, N. and J. Seward (2007). Valgrind: a framework for heavyweight dynamic binary instrumentation. *ACM Sigplan notices 42*(6), 89–100.

Nori, H., S. Jenkins, P. Koch, and R. Caruana (2019). Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks 12*(1), 145–151.

R Core Team (2021). *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rügamer, D., S. Brockhaus, K. Gentsch, K. Scherer, and S. Greven (2018). Boosting factor-specific functional historical models for the detection of synchronization in bioelectrical signals. *Journal of the Royal Statistical Society: Series C (Applied Statistics) 67*(3), 621–642.

Rügamer, D. and S. Greven (2020). Inference for L2-Boosting. *Statistics and Computing 30*, 279–289.

Saintigny, P., L. Zhang, Y.-H. Fan, A. K. El-Naggar, V. A. Papadimitrakopoulou, L. Feng, J. J. Lee, E. S. Kim, W. K. Hong, and L. Mao (2011). Gene expression profiling predicts the development of oral cancer. *Cancer Prevention Research 4*(2), 218–229.

Schalk, D., J. Thomas, and B. Bischl (2018). compboost: Modular Framework for Component-wise Boosting. *Journal of Open Source Software 3*(30), 967.

Thomas, J., S. Coors, and B. Bischl (2018). Automatic gradient boosting. In *International Workshop on Automatic Machine Learning at ICML*.

Thomas, J., T. Hepp, A. Mayr, and B. Bischl (2017). Probing for sparse and fast variable selection with model-based boosting. *Computational and mathematical methods in medicine 2017*.

Ushey, K., J. Allaire, and Y. Tang (2020). *reticulate: Interface to 'Python'.* R package version 1.18.

Wang, B., T. M. Nguyen, A. L. Bertozzi, R. G. Baraniuk, and S. J. Osher (2020). Scheduled restart momentum for accelerated stochastic gradient descent. *arXiv preprint arXiv:2002.10583*.

Wood, S. N., Z. Li, G. Shaddick, and N. H. Augustin (2017). Generalized additive models for gigadata: modeling the UK black smoke network daily data. *Journal of the American Statistical Association 112*(519), 1199–1210.

Supplementary Material for Accelerated Component-wise Gradient Boosting using Efficient Data Representation and Momentum-based Optimization

Daniel Schalk, Bernd Bischl and David Rügamer
Department of Statistics, LMU Munich

A Further Theoretical Details

A.1 Vanilla CWB Algorithm

Algorithm 1 Original CWB algorithm given the input and output.

```
    Input Train data $\mathcal{D}$, learning rate $\nu$, number of boosting iterations $M$, loss function
          $L$, set of base learner $\mathcal{B}$
    Output Prediction model $\hat{f}^{[M]}$ defined by fitted parameters $\hat{\boldsymbol{\theta}}^{[1]}, \dots, \hat{\boldsymbol{\theta}}^{[M]}$
 1: procedure CWB($\mathcal{D}, \nu, L, \mathcal{B}$)
 2:     Initialize: $\hat{f}^{[0]}(\boldsymbol{x}) = \operatorname{arg\,min}_{c \in \mathbb{R}} \mathcal{R}_{\text{emp}}(c)$
 3:     for $m \in \{1, \dots, M\}$ do
 4:         $r^{[m](i)} = -\left.\frac{\partial L\left(y^{(i)}, f\left(\boldsymbol{x}^{(i)}\right)\right)}{\partial f\left(\boldsymbol{x}^{(i)}\right)}\right|_{f=\hat{f}^{[m-1]}}, \quad \forall i \in \{1, \dots, n\}$
 5:         for $k \in \{1, \dots, K\}$ do
 6:             $\hat{\boldsymbol{\theta}}^{[m]} = \operatorname{arg\,min}_{\boldsymbol{\theta} \in \mathbb{R}^{d_k}} \sum_{i=1}^n \left(r^{[m](i)} - b_k(\boldsymbol{x}^{(i)}, \boldsymbol{\theta})\right)^2$
 7:             $\text{SSE}_k = \sum_{i=1}^n (r^{[m](i)} - b_k(\boldsymbol{x}^{(i)}, \hat{\boldsymbol{\theta}}^{[m]}))^2$
 8:         end for
 9:         $k^{[m]} = \operatorname{arg\,min}_{k \in \{1, \dots, K\}} \text{SSE}_k$
10:         $\hat{f}^{[m]}(\boldsymbol{x}) = \hat{f}^{[m-1]}(\boldsymbol{x}) + \nu b_{k^{[m]}}(\boldsymbol{x}, \hat{\boldsymbol{\theta}}^{[m]})$
11:     end for
12:     return $\hat{f} = \hat{f}^{[M]}$
13: end procedure
```

A.2 Unbiased Feature Selection

CWB also can incorporate penalties into each base learner. A bigger penalization could lead to inflexible base learners having a disadvantage in being able to fit the pseudo residuals compared to other base learners. This leads to a preference of selecting flexible base learner more often than inflexible ones. Hence, it is desirable to allow for a fair selection between more and less flexible base learners (Hofner et al., 2011). Let b_k be a penalized regression model base learner with design matrix $\boldsymbol{Z}_k$, penalty matrix $\boldsymbol{D}_k \in \mathbb{R}^{d_k \times d_k}$, an optional diagonal weight matrix $\boldsymbol{W}_k \in \mathbb{R}^{n \times n}$ as well as an optional smoothing parameter $\lambda \in \mathbb{R}$. Additionally, we assume that $\boldsymbol{W}_k$ and $\boldsymbol{D}_k$ are symmetric and can be decomposed into $\boldsymbol{W}_k = (\boldsymbol{W}_k^{1/2})^\mathsf{T}\boldsymbol{W}_k^{1/2}$ and $\boldsymbol{D}_k = (\boldsymbol{D}_k^{1/2})^\mathsf{T}\boldsymbol{D}_k^{1/2}$. The reason for that decomposition becomes clear when solving the objective to fit penalized base learner to the pseudo residuals:

$$\hat{\boldsymbol{\theta}}^{[m]} = \underset{\boldsymbol{\theta}\in\mathbb{R}^{d_k}}{\mathsf{arg\,min}} \left(\left\| \boldsymbol{r}^{[m]} - \boldsymbol{W}_k^{1/2}\boldsymbol{Z}_k\boldsymbol{\theta} \right\|_2^2 + \lambda \left\| \boldsymbol{D}_k^{1/2}\boldsymbol{\theta} \right\|_2^2 \right). \tag{1}$$

Here, $\|\cdot\|$ denotes the Euclidean norm. The fitted pseudo residuals $\hat{\boldsymbol{r}}_k^{[m]} \in \mathbb{R}^n$ are given by

$$\hat{\boldsymbol{r}}_k^{[m]} = \boldsymbol{Z}_k(\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{W}_k\boldsymbol{Z}_k + \lambda\boldsymbol{D}_k)^{-1}\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{W}_k\boldsymbol{r}^{[m]} = \boldsymbol{H}_k(\lambda)\boldsymbol{r}^{[m]} \tag{2}$$

where the decompositions $\boldsymbol{D}_k^{1/2}$ and $\boldsymbol{W}_k^{1/2}$ are aggregated to $\boldsymbol{D}_k$ and $\boldsymbol{W}_k$.

Common examples for $\boldsymbol{D}_k^{1/2}$ is the identity matrix $\boldsymbol{D}_k^{1/2} = I_n \in \mathbb{R}^{d_k \times d_k}$ used for the ridge regression or the second order difference matrix $\boldsymbol{D}_k^{1/2}(i) = (0, \dots, 0, 1, -2, 1, 0, \dots, 0)$, $\boldsymbol{D}_k^{1/2} \in \mathbb{R}^{d_k - 2 \times d_k}$, with $i-1$ zeros before 1, -2, and 1, used for P-splines. Common examples for $\boldsymbol{W}_k$ is using observational weights on the diagonal $\boldsymbol{W}_k = \mathsf{diag}\left(w_k^{(1)}, \dots, w_k^{(n)}\right)$ and $\boldsymbol{W}_k^{1/2} = \mathsf{diag}\left(\sqrt{w_k^{(1)}}, \dots, \sqrt{w_k^{(n)}}\right)$ or correcting for heteroscedasticity with $\boldsymbol{W}_k = \mathsf{diag}(1/\sigma_1^2, \dots, 1/\sigma_n^2)$ and $\boldsymbol{W}_k^{1/2} = \mathsf{diag}(1/\sigma_1, \dots, 1/\sigma_n)$ correcting for an observational based variance.

In order to ensure an unbiased selection of base learners of different complexity, the degrees of freedom of the j-th base learner fit $\hat{\boldsymbol{r}}_k^{[m]}$ are used and set equally for each base learner. Two common versions of degrees of freedom are defined via the trace of the hat matrix $\boldsymbol{H}_k(\lambda)$ (see, e.g., Buja et al., 1989):

$$\mathrm{df}_1 = \mathrm{tr}(\boldsymbol{H}_k(\lambda)), \tag{3}$$
$$\mathrm{df}_2 = \mathrm{tr}(2\boldsymbol{H}_k(\lambda) - \boldsymbol{H}_k(\lambda)\boldsymbol{H}_k(\lambda)). \tag{4}$$

The one-to-one relationship allows to define equal flexibility of each base learner in practice by defining the respective penalization such that all base learners have the same degrees of freedom. In general, there is no analytic solution for solving (3) or (4) for λ and the problem must be solved numerically. A naive uniroot search is in many cases too expensive since λ is part of the inverse matrix $(\boldsymbol{Z}_k^\mathsf{T}\boldsymbol{W}_k\boldsymbol{Z}_k + \lambda\boldsymbol{D}_k)^{-1}$ in (2) which would then have to be calculated for every λ value anew. To avoid this, the Demmler-Reinsch-Orthogonalization (DRO; Ruppert, 2002) is used.

Flexibility considerations. Choosing equal degrees of freedom might seem restrictive as some additive components naturally require more flexibility (e.g., EKG data) than others (e.g., growth curves). While a better performing model can certainly be found when tuning the degree values of all non-linear model terms, this is not only impractical but also unnecessary in many cases. Due to the greedy variable selection of CWB, additive terms that require more degrees of freedom will automatically be selected more often during the fitting process. On the other hand, the early stopping of boosting will ensure that terms with less required flexibility will get selected less often. Therefore, by setting a common, sufficiently large degree of freedom for all splines, the algorithm will automatically select covariates with a more rough signal more often and thereby allow for enough flexibility, while covariates with smoother effects will naturally be selected less frequent and thereby subject to more shrinkage.

A.3 Categorical Features

Categorical features $\boldsymbol{x}_j$ consisting of c_j classes $\{1, \dots, c_j\}$. These classes do not necessarily need to be encoded as integer values, but we keep this formalization due to simplicity and unified notation. For example, a categorical feature for three different prescribed medications consists of three classes $1 =$ *Medication 1*, $2 =$ *Medication 2*, and $3 =$ *Medication 3*.

The basic idea of efficient categorical feature representations is to make use of the number of classes c_j which is much smaller than the number of observations n. Working on the number of classes allows us to fit base learner of categorical features faster and with less memory usage than operating on the instances. We describe two common representation for CWB, highlight their advantages, and how to interpret the estimated base learner.

A.3.1 Ridge Representation

One possibility to include categorical features into a base learner is to make use of techniques applied in linear models. The usual way to encode categorical features is to use a dummy-encoding represented by a binary model matrix $\boldsymbol{X}_j \in \mathbb{R}^{n \times c_j}$ with elements $\boldsymbol{Z}_j(i,k) = \mathbb{1}_{\{x_j^{(i)}=k\}}$, $k \in \{1, \dots, c_j\}$. To obtain the regression coefficients of a base learner with categorical feature $\boldsymbol{x}_j$, we use ridge regression (Hoerl and Kennard, 1970) and solve

$$\hat{\boldsymbol{\theta}}^{[m]} = \operatorname*{arg\,min}_{\boldsymbol{\theta} \in \mathbb{R}^{c_j}} \|\boldsymbol{r}^{[m]} - \boldsymbol{X}_j\boldsymbol{\theta}\|_2^2 + \lambda\|\boldsymbol{\theta}\|_2^2 \tag{5}$$

where $\lambda \in \mathbb{R}_+$ is a penalty which additionally shrinks the coefficients. Equation (5) can be solved analytically with solution $\hat{\boldsymbol{\theta}}^{[m]} = (\boldsymbol{Z}_j^\mathsf{T}\boldsymbol{Z}_j + \lambda\boldsymbol{I}_{c_j})^{-1}\boldsymbol{Z}_j^\mathsf{T}\boldsymbol{r}^{[m]}$ where $\boldsymbol{I}_{c_j} \in \mathbb{R}^{c_j \times c_j}$ is the identity matrix. Advantages of this representation are:

Efficient fitting procedure that analytically calculates the inverse of $\boldsymbol{X}_j^\mathsf{T}\boldsymbol{X}_j + \lambda\boldsymbol{I}_{c_j}$ by making use of the diagonal structure. The inverse is calculated as $\mathsf{diag}((n_{j,1} +$

$\lambda)^{-1}, \dots, (n_{j,c_j} + \lambda)^{-1})$ where the elements $n_{j,k} = \sum_{i=1}^n \mathbb{1}_{\{x_j^{(i)} = k\}}$ correspond to the cardinality of class k of the j-th feature. Instead of storing the binary matrix $\boldsymbol{X}_j$ which requires at least $2n$ integer values for a sparse data format, we store c_j double values representing the diagonal. Additionally, we do not have to calculate the inverse in each iteration. Furthermore, the matrix multiplication $\boldsymbol{u} = \boldsymbol{X}_j \boldsymbol{r}^{[m]}$ can be efficiently calculated by group means $u_k = \sum_{i=1}^n \mathbb{1}_{\{x_{i,j} = k\}} r^{[m](i)}$, $k \in \{1, \dots, c_j\}$.

Explicit calculation of the degrees of freedom to conduct an unbiased feature selection between base learners even if the base learners are not from the same type (e.g. comparison of spline and ridge base learner). The calculation is done by making use of the diagonal structure. As mentioned in Section A.2 we have to use the DRO which in turn requires a *singular value decomposition* (SVD) to calculate the degrees of freedom. Because of the diagonal structure, it is not necessary to calculate the SVD explicitly. The degrees of freedom are directly given as function of the number of observations:

$$\mathrm{df}_1 = \sum_{k=1}^{c_j} \frac{n_{j,k}}{(n_{j,k} + \lambda)} \tag{6}$$

$$\mathrm{df}_2 = \sum_{k=1}^{c_j} \frac{n_{j,k}(n_{j,k} + 2\lambda)}{(n_{j,k} + \lambda)^2}. \tag{7}$$

Number of base learner to loop over is independent from the number of classes. Using the ridge representation constructs one base learner per categorical feature. Therefore, the number of base learners and the complexity of the model does not increase with increasing numbers of classes c_j as it is for the binary representation in the next section.

A.3.2 Sparse binary Representation

Instead of having one base learner per feature, another representation can be applied on the class level. Therefore, each class $k \in \{1, \dots, c_j\}$ of the categorical feature $\boldsymbol{x}_j$ is a new base learner containing a binary vector just for that class. The model matrix $\boldsymbol{X}_j$ then is of dimension $n \times 1$ and contains ones at the respective entries and otherwise zeros. The parameter estimate is again calculated using the method of least squares

$$\hat{\theta}_{j,k}^{[m]} = (\boldsymbol{X}_j^\mathsf{T} \boldsymbol{X}_j)^{-1} \boldsymbol{X}_j^\mathsf{T} \boldsymbol{r}^{[m]} = n_{j,k}^{-1} \sum_{i=1}^n \mathbb{1}_{\{x_{i,j} = g\}} r^{[m](i)}. \tag{8}$$

Following this definition, the coefficient $\hat{\theta}_{j,k}$ is a scalar representing the k-th class mean of feature j. Advantages of this representation are as follows:

Automated class selection within the categorical feature. Compared to updating mechanisms where all parameter are updated at once (i.e., as for the ridge representation) the binary representation selects just one class per iteration. This yields an automatic class selection induced by the fitting process. Further information about importance of classes can then be derived from the trace on how the classes are selected. This is also helpful when one of the classes does not contain information and therefore should not be selected.

A sparser model is obtained due to individual class selections within one categorical feature.

B Further Experimental Findings and Details

B.1 Robustness of Binning

In order to illustrate the robustness of binning in challenging data setups, we conduct two simulation studies in the following. Our first study investigates the behavior of binning when the feature distribution is (highly) skewed, the second study examines the influence of outliers on binning.

B.1.1 Binning with Skewed Feature Distribution

We first study the application of binning on features with a (highly) skewed distribution and its influence on the estimation quality of feature effects. We therefore sample a feature x using a Beta(α, β) distribution and simulate its non-linear effect as explained in Section 4.1.1. To simulate differently skewed feature distributions, we vary the Beta distribution's shape parameters using $\alpha \in \{1, 3, 5, 7, 9\}$ and $\beta \in \{1, 3, 5, 7, 9\}$. For each setting we draw $n \in \{500, 1000, 5000, 10000\}$ observations and repeat the experiment 50 times. Figure 1 exemplary shows the Beta distribution for four different (α, β) values as well as the empirical skewness measured by the method of moments, i.e., $n^{-1}\sum_{i=1}^{n}(x^{(i)} - \bar{x})^3/((n-1)^{-1}\sum_{i=1}^{n}(x^{(i)} - \bar{x})^2)^{3/2}$ with average feature value $\bar{x}$.

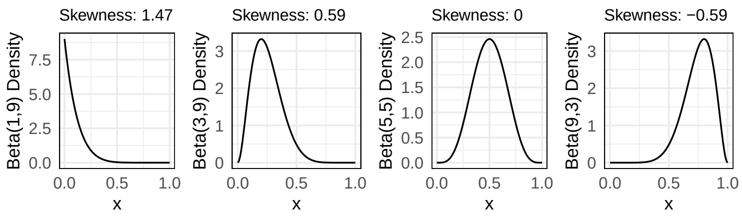

Figure 1: Densities for different beta distributions.

We train CWB with and without binning and compare the two models' estimation performance using the MISE as in Section 4.1.1. Figure 2 visualizes the relative change in MISE value when using binning compared to no binning (i.e., subtracting the MISE when using binning from the MISE when using no binning and dividing by the MISE without binning). A value of -0.1, e.g., indicates that binning performs 10% worse than no binning. While larger deviations from zero can be observed, the relative change in MISE fluctuates around zero with no particular trend in one direction. Based on these results we expect binning to be robust for skewed feature distributions.

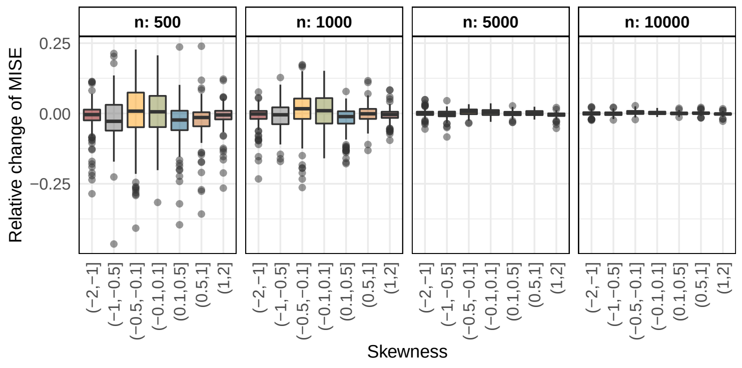

Figure 2: Relative change of the MISE for different data sizes and skewed distributions.

B.1.2 Binning with Outliers

To investigating the effect of outliers in features, we simulate a feature x and set $n \cdot q$ random feature values to extreme values, where $q \in [0, 1)$ represents the proportion of outliers in the data. Outlier values are defined by $\max_i(x^{(i)}) + m_{\text{sd}} \cdot \sigma_x + \varepsilon$, where $\max_i(x^{(i)})$ is the empirical maximum of all simulated feature values without outliers, σ_x the empirical standard deviation of values, m_{sd} the factor that controls the distance of outliers to the maximum value, and $\varepsilon \sim \mathcal{N}(0, \sigma_x)$ a normal distributed additional variation. We use a Beta$(2, 2)$ distribution (which has a true standard deviation of around 0.22). We investigate outlier factors $m_{\text{sd}} \in \{4, 16, 64\}$, with amounts $q \in \{0.01, 0.05, 0.1\}$, and $n \in \{500, 1000, 5000, 10000\}$. While non-outlier feature values have values between 0 and 1, outliers have a value of ≈ 2 for $m_{\text{sd}} = 4$, ≈ 4.5 for $m_{\text{sd}} = 16$, and ≈ 14 for $m_{\text{sd}} = 64$. As in the previous simulation study, we compare the estimation performance of CWB with and without binning by computing the relative change in MISE (visualized in Figure 3) by repeating every experiment 50 times. We observe that for smaller number of observations ($n = 500$) and $m_{\text{sd}} = 64$, binning shows notably worse estimation performance compared to no binning. For increasing n, the estimation performance of CWB with binning improves as the number of bins also increases and becomes sufficiently large. While our results show that binning fails for smaller data sets, we want to highlight that binning is primarily used in big data settings as the runtime and memory improvement are negligible for small data sets.

B.2 Simulation Study - Categorical Features

We do not compare the two categorical encodings explained in this paper since there is no state-of-the-art encoding to compare with. Instead, we want to highlight their computational properties and also how to interpret the estimated base learner with the respective encoding.

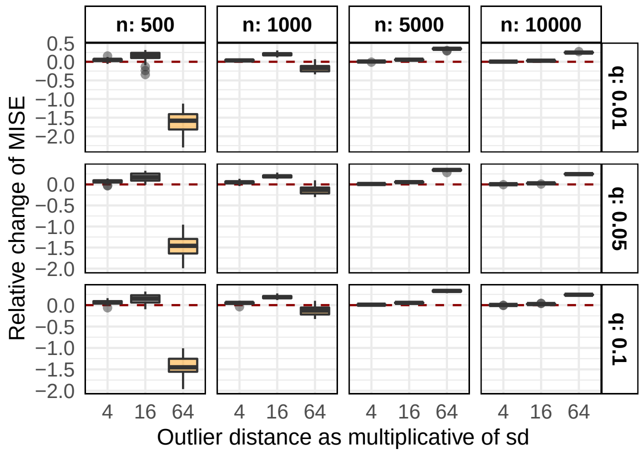

Figure 3: Relative change of MISE values for different data sizes, outlier proportions and the outlier distance measured in multiples of the features standard deviation.

As shown in Figure 4, the ridge encoding is much faster in terms of runtime and memory. The effect can be explained by again looking at the structure of the base learners. Using the binary encoding requires to fit as many base learners as classes in the feature. The high memory consumption when using binary base learner can be explained by looking at the metadata the base learner stores. Each binary base learner holds a vector of indexes to calculate the group mean in each iteration. The reason for that is to not loop over all n feature values in each iteration for each binary base learner but just over the subset of feature values corresponding to the specific class. In contrast, the ridge base learner stores one vector with the number of observations per class and uses the original feature value to calculate the group means.

The binary encoding surpasses the ridge encoding in terms of filtering non-informative classes. As we can see in Figure 5, the binary encoding is able to filter non-informative classes while the ridge representation does not filter them. Note that the points for the ridge encoding are all placed at $(0, 0)$ due to the simultaneous updating of all class parameter. In contrast, the binary representation updates just one class parameter per iteration. The cost for a better TNR comes with the risk of not selecting important features (higher FNR) for a higher SNR. Hence, the binary encoding has a very conservative selection process, but if a class is selected it was likely selected due to a signal present in the data.

As shown in Figure 6, the MSE of the estimated parameter and the true ones is small for a smaller SNR but increases for more complex situations. The MSE of the binary encoding is slightly better as for the ridge representation but is slightly worse for the high SNR case. Even though the small MSE of the parameter of non-informative classes, they appear

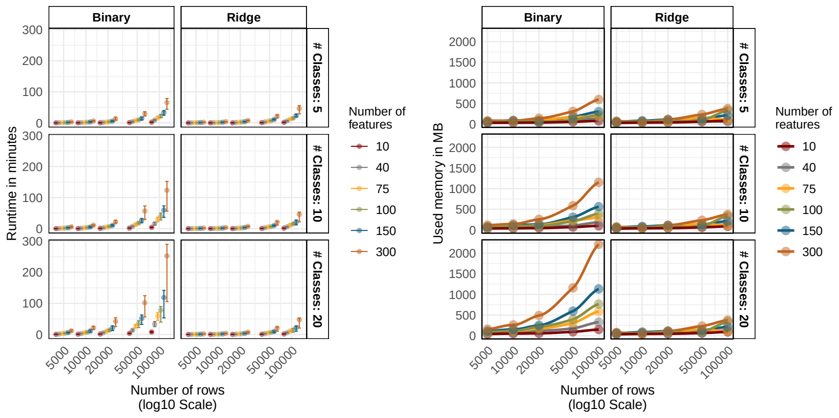

Figure 4: Runtime in seconds (left). Memory consumption in megabyte (right).

in the model and therefore increase the complexity. This is especially the case for ridge regression. A strategy to overcome this issue is to set a threshold for which the smaller parameters are set to zero. As we can see in Figure 6, this cutoff does not affect the MSE but leads to a better TNR than ridge regression (see Figure 5). The challenge now is to accordingly set this cutoff value to not cut parameters too aggressively and therefore ignore informative classes. Figures 5 and 6 contains this strategy for a cutoff of 0.01, 0.5, and 1. As we can see, the TNR is improves while the FNR gets worse. Looking at Figure 6 in the context of Figure 5, we can explain the higher MSE for binary base learner for a SNR of 0.1 due to the higher FNR and therefore a higher MSE on the not selected classes.

To summarize the results, it is much faster and more memory friendly to use the ridge representation. The drawback of the ridge representation is the risk of wrongly selecting non-informative classes. Despite the fact that we can cut off classes, the way binary encoding selects classes is more natural. If computational resources are an issue, we suggest to use the ridge representation while the binary representation gives us a sparser model and a more precise selection of classes.

B.3 Modelling Big Data with Binning

One of the primary goals of using binning is to scale CWB to big data applications. Although computational resources become cheaper and cheaper, fitting models on larger data sets is still infeasible with common laptops or smaller servers. We showcase this by fitting CWB to three large data sets that are publicly available, using a machine with 32 GB of RAM. The data sets are:

- The HIGGS data set (Baldi et al., 2014) known from physics and available in the UCI Machine Learning Repository (Dua and Graff, 2017). The size of the data set is 2.4

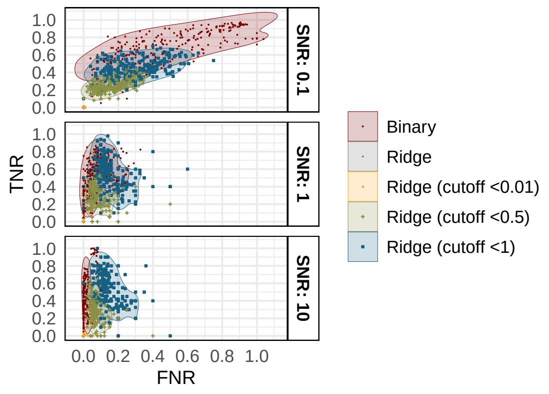

Figure 5: Fraction of being able to filter non-informative classes (TNR; class was not selected and true parameter is zero) and wrongly filtering informative classes (FNR; class was not selected but true parameter is not zero). One point in the figure corresponds to the median of the 20 replications of one configuration. The optimal point would be in the left upper corner at $(0, 1)$, meaning that all non-informative classes were filtered and all informative classes were selecting all classes with a signal. The contour lines are the two dimensional empirical 0.95 quantiles.

GB. It contains $n = 11 \times 10^6$ observations and 29 numerical features. The goal is to distinguish between a signal process which produces Higgs bosons and a background process.

- The NYC Yellow Taxi Trip data set contains taxi rides in New York. The data set is available on Kaggle[1] (Version 2) with rides from January to March 2016. The original data is publicly at `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`. The goal is to predict the total amount of costs for a taxi ride. The data set contains $n = 24.3 \times 10^6$ observations with 22 features from which 22 are numeric. The data set size is 3.3 GB.

- FEMA's National Flood Insurance Policy Database contains national flood insurance policy transactions. For our demonstration, we use properties built between 1990 and 2010. The data set is available on Kaggle[2] (Version 2). The original data are available at `https://www.fema.gov/about/reports-and-data/openfema`. The goal of this task is to predict the total insurance premium of a policy. The data set contains $n = 14.5 \times 10^6$ observations and 50 features from which 29 are numeric. The size of the data set is 3.4 GB.

For each data set we choose a spline base learner with 20 basis functions for every numerical feature. As most of the memory is allocated during the initialization of CWB, we train

[1] `https://www.kaggle.com/datasets/elemento/nyc-yellow-taxi-trip-data`
[2] `https://www.kaggle.com/datasets/lynma01/femas-national-flood-insurance-policy-database`

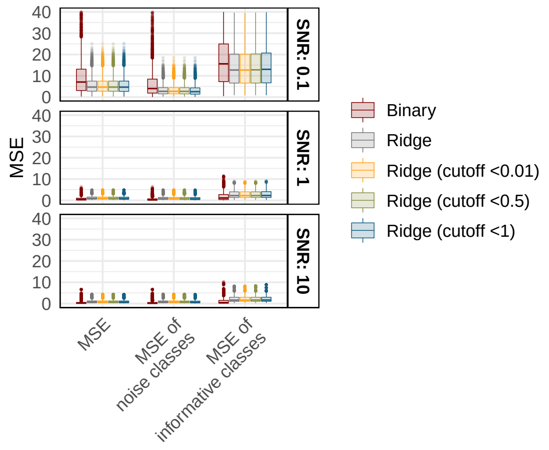

Figure 6: MSE of the estimated parameter the true ones. The MSE is shown for all parameter (MSE), parameters corresponding to noise classes, and parameter corresponding to informative classes (MSE of informative classes)

the model for just 50 iterations using the above data sets. In order to investigate the differences in memory consumption we train the model twice using the exact same setup, one time with binning and once without binning. Figure 7 visualizes the traces of allocated memory during the data import and the model fitting phase. For all data sets, CWB can be successfully used to fit the model when binning is activated. Without binning, the initialization process allocates too much memory and the CWB routine crashes.

B.4 Algorithms in Benchmark

Algorithm	Software	Hyperparameter space
CWB/ACWB/HCWB (nb) CWB/ACWB/HCWB (b)	`compboost`	`df` $\in [2, 10]$ `df_cat` $\in [2, 10]$ `learning_rate` $\in [0.001, 0.5]$
XGBoost	`xgboost` (Chen et al., 2018)	`eta` $\in [0.001, 0.5]$ `max_depth` $\in \{1, \ldots, 20\}$ `colsample_bytree` $\in [0.5, 1]$ `colsample_bylevel` $\in [0.5, 1]$ `subsample` $\in [0.3, 1]$ `lambda` $\in \{2^\lambda \mid \lambda \in [-10, 10]\}$ `alpha` $\in \{2^\alpha \mid \alpha \in [-10, 10]\}$
EBM	`interpret` (Nori et al., 2019)	`learning_rate` $\in [0.001, 0.5\}$

Table 1: Algorithm name, software package, HP space, and number of outer evaluations of all modeling techniques compared in the benchmark.

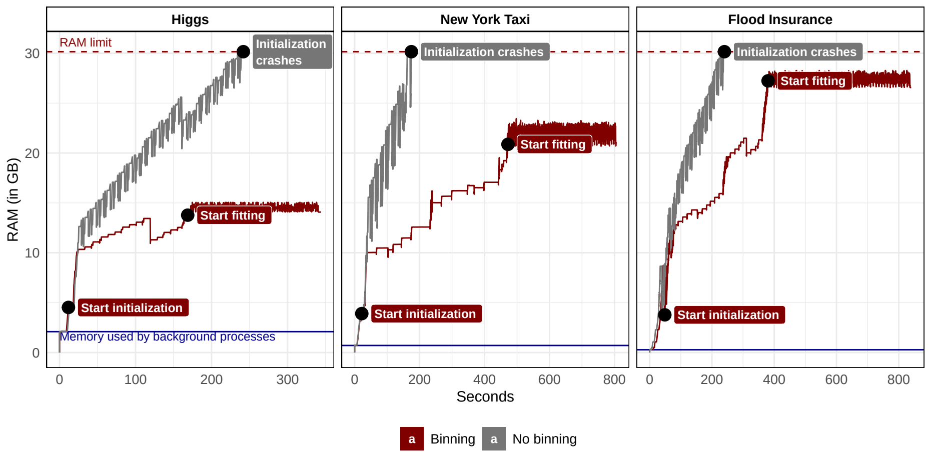

Figure 7: Memory traces of CWB with and without binning for the three data sets. The memory limit (32 GB) is depicted as a dashed horizontal line. The fitting process of CWB crashes once the trace of the allocated memory reaches this limit.

Table 1 gives an overview of all methods used in our benchmarks and their corresponding hyperparameter spaces.

B.5 Used Real World Data Sets

Data sets: We use 6 data sets from OpenML (Vanschoren et al., 2013; Casalicchio et al., 2017) for binary classification and provide descriptive statistics in Table 2. The data sets are pre-processed by imputing missing values in numerical features with the median, in categorical features with the mode, and then removing constant features from the data[3].

B.6 Picture of varying Degrees of Freedom and Number of Bins

To get further insights how the choice of number of bins affects the performance, we conduct the benchmark with $n^* = n^{1/4}$. Additionally, we choose different values for the degrees of

[3]Note that we keep thing simple here by applying all preprocessing to the complete data sets, which does not affect the validity of our comparison experiments. In proper applied work, such pre-processing should be embedded into cross-validation.

Data set	Data ID	# Samples	# Features	
			Numeric	Categorical
Spam	44	4601	57	0
Christine	41142	5418	1599	37
Namao	1486	34465	83	29
Adult	1590	48842	6	8
MiniBooNE	41150	130064	50	0
Albert	41147	425240	26	52

Table 2: Key characteristics of the used data sets.

freedom. The results are shown in Figure 8.

B.7 Empirical Assessment of Computational Complexity

B.7.1 Binning

Modeling the time as proxy for the computational complexity as calculated in Section 2.4 for CWB and 3.1.3 for CWB with binning confirmes our complexity estimates with $\mathcal{O}(K^{0.989}(d^{2.014}n^{1.030}+d^{2.985})+M^{1.025}K^{1.006}(d^{2.040}+d^{0.992}n^{0.982}))$ for CWB and respectively $\mathcal{O}(K^{1.034}(d^{1.989}(n^*)^{0.994}+n^{0.956}+d^{3.065})+M^{1.020}K^{0.992}(d^{2.000}+d^{1.005}(n^*)^{0.998}+n^{0.993}))$ for CWB with binning. Figure 9 shows the fitted curves. Additionally, a scale parameter $v = 3.989 \cdot 10^7$ for CWB and $v = 1.072 \cdot 10^7$ for CWB with binning was estimated to account for different scales of seconds and number of operations. The R-square of the fitted curves is 1 for CWB and 0.999 for CWB with binning.

B.7.2 ACWB

Following our derivations in Section 3.2.2, we expect the runtime of ACWB to scale linear with the number of observations n as well as the number of base learners K. Therefore, we model the time as proxy for the computational complexity. The fit confirms our complexity estimates with $\mathcal{O}(K^{0.861}(d^{2.165}n^{0.835}+d^{2.742})+2M^{0.769}K^{1.002}(d^{1.904}+d^{1.134}n^{1.087}))$. Figure 10 shows the fitted curves. Additionally, a scale parameter $v = 2.273 \cdot 10^7$ was estimated to account for different scales of seconds and number of operations. The R-square of the fitted

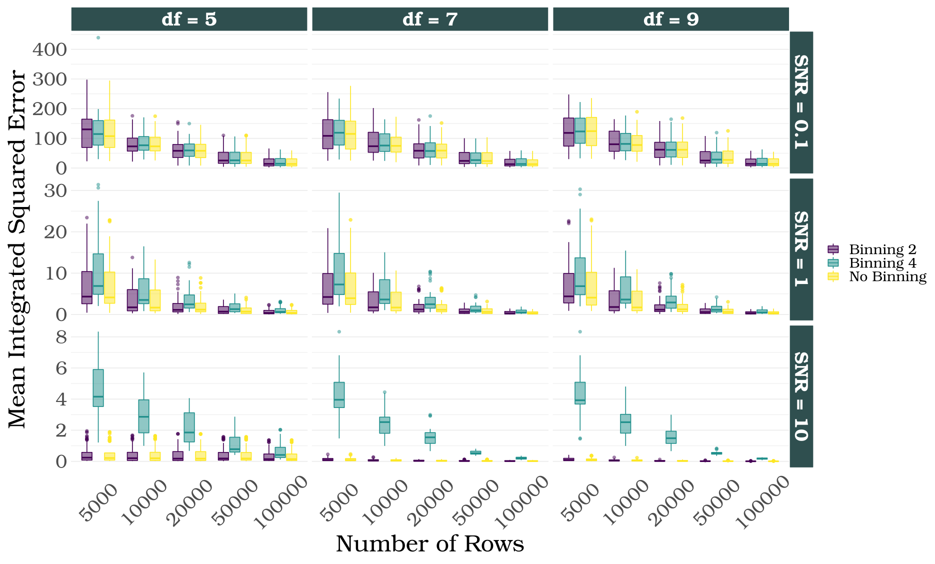

Figure 8: MISE of binning when applied with $n^* = n^{1/2}$ and $n^* = n^{1/4}$ as well as df $\in \{5, 7, 9\}$.

curves is 0.9779. As shown, ACWB scales efficiently with $\mathcal{O}(n)$ and $\mathcal{O}(K)$. Considering the complexity of HCWB, it is sufficient to know $\mathcal{O}(\text{HCWB}) = a\mathcal{O}(\text{CWB}) + b\mathcal{O}(\text{ACWB}) = \mathcal{O}(\text{ACWB})$. It is worth noting that the empirical and theoretical claims do not show how fast the algorithms are in practice due to the use of early stopping procedures as well as the simplification of just considering numerical features with $d_k = d$ and using a fixed number of iterations.

B.8 Hyperband Schedule

Table 3 contains the schedule used for all algorithms to tune the HPs.

B.9 EQ1: Figure of comparing `compboost` with `mboost`

Figure 11 visualizes the speedup of CWB/ACWB/HCWB implemented in `compboost` when compared with the CWB implementation of `mboost`. It should be noted that there is no

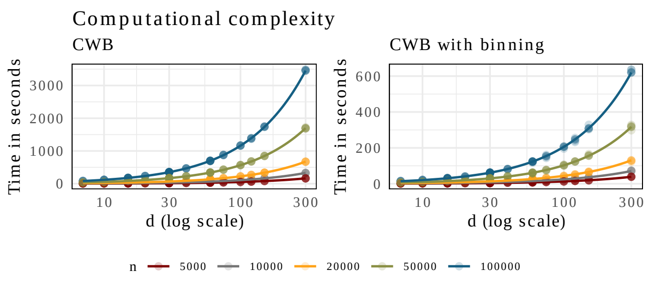

Figure 9: Fitted curves based on computational complexity with time used as proxy and response variable. The number of rows and base learner are used as input.

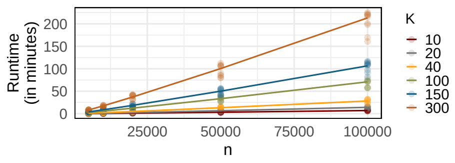

Figure 10: Fitted curves based on computational complexity with fitting time of ACWB used as proxy and response variable. The number of rows n and base learners K are used as input.

data available for the Christine data set since it was not possible to train CWB using `mboost` for this data set.

B.10 EQ2: Relative Speedup and AUC Improvement

Table 4 shows the speedup of our CWB variants when compared with vanilla CWB.

B.11 EQ2: Summary of GLM and ANOVA on the AUC

Conducting the GLM with the AUC as response contains the task, learner (CWB variant), and a binary variable if binning was applied. Additionally, interactions between binning and the learner are included. The model formula is given by

$$auc \sim task + binning + optimizer + binning * learner.$$

Table 5 and 6 contains the `R` output of the beta regression and the corresponding ANOVA.

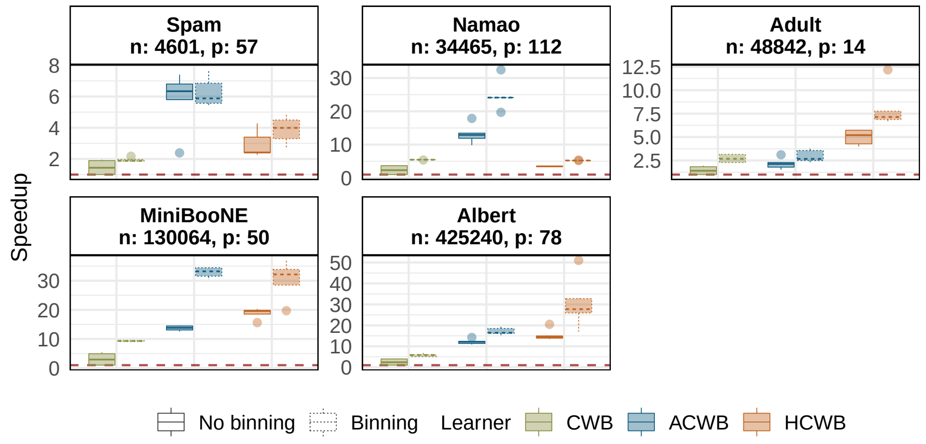

Figure 11: Speedup of our CWB variants compared to the state-of-the-art implementation `mboost`. Values on the y-axis visualize the speedup calculated as the runtime of `mboost` divided by the runtime of our implementations.

B.12 EQ3: Table of the Renchmark Result

References

Baldi, P., P. Sadowski, and D. Whiteson (2014). Searching for exotic particles in high-energy physics with deep learning. *Nature communications 5*(1), 1–9.

Buja, A., T. Hastie, and R. Tibshirani (1989). Linear smoothers and additive models. *The Annals of Statistics*, 453–510.

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

Chen, T., T. He, M. Benesty, V. Khotilovich, and Y. Tang (2018). *xgboost: Extreme Gradient Boosting*. R package version 0.6.4.1.

Dua, D. and C. Graff (2017). UCI machine learning repository.

Hoerl, A. E. and R. W. Kennard (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics 12*(1), 55–67.

Hofner, B., T. Hothorn, T. Kneib, and M. Schmid (2011). A framework for unbiased model selection based on boosting. *Journal of Computational and Graphical Statistics 20*(4), 956–971.

Nori, H., S. Jenkins, P. Koch, and R. Caruana (2019). Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*.

Ruppert, D. (2002). Selecting the number of knots for penalized splines. *Journal of computational and graphical statistics 11*(4), 735–757.

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

Bracket	Stage	Budget	#HP configurations
7	0	39	128
7	1	78	64
7	2	156	32
7	3	312	16
7	4	625	8
7	5	1250	4
7	6	2500	2
7	7	5000	1
6	0	78	74
6	1	156	37
6	2	312	18
6	3	625	9
6	4	1250	4
6	5	2500	2
6	6	5000	1
5	0	156	43
5	1	312	21
5	2	625	10
5	3	1250	5
5	4	2500	2
5	5	5000	1
4	0	312	26
4	1	625	13
4	2	1250	6
4	3	2500	3
4	4	5000	1
3	0	625	16
3	1	1250	8
3	2	2500	4
3	3	5000	2
2	0	1250	11
2	1	2500	5
2	2	5000	2
1	0	2500	8
1	1	5000	4
0	0	5000	8

Table 3: The schedule used by Hyperband for HP optimization. The total number of tried HP configuration (314) is given by accumulating the number of HP configurations in each bracket at stage 0.

Learner /	**Spam**		**Christine**		**Namao**		**Adult**		**MiniBooNE**		**Albert**	
Binning	Δt	dAUC	Δt	dAUC	Δt	dAUC	Δt	dAUC	Δt	dAUC	Δt	dAUC
CWB (b)	1.014	0.000	2.003	-0.002	1.482	0.000	1.363	0.000	1.866	0.000	1.375	0.000
ACWB (nb)	3.147	-0.013	9.072	0.003	3.505	-0.016	1.173	0.001	2.893	0.003	2.963	0.002
ACWB (b)	3.277	-0.012	15.476	-0.001	6.554	-0.012	1.500	0.001	6.435	0.003	3.950	0.001
HCWB (nb)	1.451	-0.005	2.008	-0.009	0.940	-0.003	2.582	0.001	3.936	0.002	3.388	0.001
HCWB (b)	1.989	-0.006	1.502	0.001	1.422	-0.004	3.856	0.001	6.627	0.002	7.547	0.000

Table 4: Relative speedup (Δt) and AUC improvement (dAUC) of the 5 CWB variants compared to CWB (nb). The AUC improvement is calculated as $\text{AUC}_{\text{learner}} - \text{AUC}_{\text{CWB (nb)}}$.

	Estimate	Std. Error	z value	$Pr(>\|z\|)$
(Intercept)	3.5672	0.0310	115.1075	0.0000
task168908	-2.1910	0.0297	-73.7020	0.0000
task7592	-1.1963	0.0319	-37.5123	0.0000
task168335	0.0456	0.0393	1.1593	0.2463
task189866	-2.5422	0.0294	-86.5780	0.0000
task9977	0.3423	0.0425	8.0538	0.0000
binningBinning	-0.0011	0.0220	-0.0510	0.9594
learnerACWB	-0.0180	0.0220	-0.8200	0.4122
learnerhCWB	-0.0087	0.0220	-0.3952	0.6927
binningBinning:learnerACWB	-0.0035	0.0311	-0.1115	0.9112
binningBinning:learnerhCWB	0.0025	0.0311	0.0796	0.9366

Table 5: Summary results of the GAM.

	df	Chi.sq	p-value
task	5	20785.3355	0.0000
binning	1	0.0026	0.9594
learner	2	0.6727	0.7144
binning:learner	2	0.0370	0.9817

Table 6: Results of the ANOVA applied on the GLM.

	Algorithm	Data set					
		Spam	**Christine**	**Namao**	**Adult**	**MiniBooNE**	**Albert**
AUC	ACWB (b)	0.969 ± 0.009	0.794 ± 0.019	0.98 ± 0.001	0.915 ± 0.002	0.974 ± 0.001	0.735 ± 0.001
	hCWB (b)	0.969 ± 0.009	0.797 ± 0.016	0.978 ± 0.001	0.916 ± 0.002	0.975 ± 0.001	0.735 ± 0.001
	EBM	0.986 ± 0.005	0.814 ± 0.012	0.994 ± 0.000	0.928 ± 0.001	0.978 ± 0.000	0.747 ± 0.000
	XGBoost	0.99 ± 0.002	0.823 ± 0.016	0.996 ± 0.000	0.929 ± 0.001	0.987 ± 0.000	0.756 ± 0.001
Runtime (in hours)	ACWB (b)	2.823 ± 0.008	17.741 ± 0.274	3.884 ± 0.009	1.665 ± 0.022	7.419 ± 0.027	24.74 ± 0.193
	hCWB (b)	2.832 ± 0.009	27.361 ± 0.266	3.978 ± 0.026	1.511 ± 0.021	7.476 ± 0.047	26.249 ± 0.228
	EBM	3.711 ± 0.103	164.228 ± 0.441	15.668 ± 0.000	8.172 ± 0.426	28.183 ± 0.751	148.927 ± 0.000
	XGBoost	0.911 ± 0.010	6.053 ± 0.582	3.829 ± 0.117	3.975 ± 0.224	15.401 ± 0.3	96.727 ± 5.029

Table 7: Average and standard deviation of AUC values as well as runtimes (in hours) for the 5 outer folds of our experiments.

CHAPTER 8

Automatic Componentwise Boosting:An Interpretable AutoML System

Contributing article

Coors, S., Schalk, D., Bischl, B., and Rügamer, D. (2021). Automatic componentwise boosting: An interpretable automl system. *ECML-PKDD Workshop on Automating Data Science*

Declaration of contributions

Daniel Schalk was significantly involved in the method development and implementation. For this, he designed the basic concept with Stefan Coors and developed large parts of the prototype. Furthermore, he refined the software and implemented visualization techniques to interpret the model. He wrote the theoretical basis for CWB and the text passages explaining the methodology in the manuscript. In addition, he helped revise all sections.

Contribution of the coauthors

Stefan Coors and Daniel Schalk were substantially involved in implementing the method. Additionally, Stefan Coors was responsible for conducting the benchmark to compare `Autocompboost` with established AutoML systems. In addition, Stefan Coors and David Rügamer assisted in fine-tuning the methodology. All co-authors were equally involved in the preparation of the manuscript.

Automatic Componentwise Boosting: An Interpretable AutoML System

Coors Stefan[1][0000−0002−7465−2146], Schalk Daniel[1][0000−0003−0950−1947], Bischl Bernd[1][0000−0001−6002−6980], and Rügamer David[1][0000−0002−8772−9202]

Department of Statistics, LMU Munich, Germany
{firstname.lastname}@stat.uni-muenchen.de

Abstract. In practice, machine learning (ML) workflows require various different steps, from data preprocessing, missing value imputation, model selection, to model tuning as well as model evaluation. Many of these steps rely on human ML experts. AutoML – the field of automating these ML pipelines – tries to help practitioners to apply ML off-the-shelf without any expert knowledge. Most modern AutoML systems like auto-sklearn, H20-AutoML or TPOT aim for high predictive performance, thereby generating ensembles that consist almost exclusively of black-box models. This, in turn, makes the interpretation for the layperson more intricate and adds another layer of opacity for users. We propose an AutoML system that constructs an interpretable additive model that can be fitted using a highly scalable componentwise boosting algorithm. Our system provides tools for easy model interpretation such as visualizing partial effects and pairwise interactions, allows for a straightforward calculation of feature importance, and gives insights into the required model complexity to fit the given task. We introduce the general framework and outline its implementation `autocompboost`. To demonstrate the frameworks efficacy, we compare `autocompboost` to other existing systems based on the OpenML AutoML-Benchmark. Despite its restriction to an interpretable model space, our system is competitive in terms of predictive performance on most data sets while being more user-friendly and transparent.

Keywords: Interpretable ML · Boosting · AutoML · Splines · Additive Models · Deep Trees · Variable Selection.

1 Introduction and Related Work

Machine learning (ML) models achieve state-of-the-art performances in many different fields of application. Their increasing complexity allows them to be adapted well to non-trivial data generating processes. However, applying ML in practice is usually accompanied with many more hurdles that require both time and expert knowledge. Challenges in the application of ML are, amongst others, proper data preprocessing, missing value imputation, and hyperparameter optimization (HPO). This so-called ML pipeline is usually non-trivial and requires a "human in the loop". AutoML systems (Gijsbers et al., 2019) such as Auto-WEKA (Kotthoff et al., 2019), Auto-sklearn (Feurer et al., 2019), or autoxgboost (Thomas et al., 2018) attempt to automate ML pipelines based on well-defined routines. Human expert knowledge is encoded into an automated process to reduce input from the end user. Automation can also be more time-efficient and superior in predictive performance. To this end, many AutoML systems do not limit themselves to one type of ML model but instead try to solve, e.g., a Combined Algorithm Selection and Hyperparameter Optimization problem (CASH; Kotthoff et al., 2019). Results of AutoML systems are thus usually large ensembles of different models. Recent frameworks like AutoGluon-Tabular (Erickson et al., 2020) and Auto-PyTorch Tabular (Zimmer et al., 2021) even incorporate deep neural networks in their systems to further increase the model complexity.

Allowing AutoML systems to build complex ensembles of models increases the likelihood of the automated system to work well on most given data sets without further user input. The complexity of these ensembles, however, makes it harder for practitioners to understand the model's decision process – an aspect of applied ML that is often at least as important as the predictive performance itself. This results in AutoML systems often not being adopted in practice (Drozdal et al., 2020) and leads researchers in the field of AutoML to raise the question of a trade-off between predictive performance and interpretability (Pfisterer et al., 2019; Freitas, 2019; Xanthopoulos et al., 2020). In this work, we hypothesize that this model complexity is in many cases unnecessary; an interpretable AutoML system performs equally well on most tasks, yet without the need to use additional interpretation tools or the uncertainty about the model's decisions.

1.1 Our Contribution

We propose a scalable and flexible solution to both automate the ML pipeline targeted by most of the existing AutoML systems and to provide an inherently interpretable model that 1) does not require post-model fitting explanation methods, 2) automatically yields all the characteristics to understand the final chosen model, while 3) yielding (close to) state-of-the-art performance on most practical use cases. Our methodological contribution based on a novel stage- and componentwise boosting algorithm is accompanied by an application as well as benchmark experiments to underline the idea and efficacy of our approach. We have implemented our AutoML system in the `R` (R Core Team, 2021) package `autocompboost` available on GitHub[1].

2 Automatic Componentwise Boosting

In the following, we explain our framework in two steps. First, Section 2.1 describes the details of our proposed algorithm to fit models based on the preprocessed data. The second part (Section 2.2) is concerned with the automation of applying the algorithm to a given task.

[1] `github.com/Coorsaa/autocompboost`

2.1 Fitting Engine

Instead of using an ensemble of different models, we propose using componentwise boosting (CWB; Bühlmann and Yu, 2003) as a fitting engine. CWB uses additive models as base learners iteratively fitted to pseudo residuals in each boosting iteration with a learning rate $\nu \in \mathbb{R}_+$. Due to the additivity of these updates and the structure of the base learners (linear models, splines, tensor-products splines), an interpretable model is obtained. Due to its componentwise nature, CWB also comes with an inherent feature selection mechanism, similar to the Lasso (Meinshausen et al., 2007). CWB can therefore be used for high-dimensional feature spaces ($n \ll p$ situations) and, when penalized properly, without a biased base learner selection towards more flexible terms (Hofner et al., 2011). Further advantages and extensions are given in Appendix A.

Instead of using the vanilla CWB algorithm, we propose a novel stagewise procedure that has all the advantages of CWB but allows practitioners to better control overfitting and provides further insights into the modeled relationships.

Stagewise model fitting We define the final model f to be a combination of three parts: *i*) univariate (linear + non-linear), *ii*) pairwise interactions, and *iii*) deep interactions, resulting in $f(\boldsymbol{x}) = f_{\text{uni}}(\boldsymbol{x}) + f_{\text{pint}}(\boldsymbol{x}) + f_{\text{deep}}(\boldsymbol{x})$, fitted in three consecutive stages. The number of boosting iterations of each stage is dynamically selected by early stopping. Hence, if no risk improvement on a validation set is observed κ consecutive times (our default: $\kappa = 2$), the fitting proceeds to the next stage.

In the first stage, we use CWB on all available features $\boldsymbol{x} \in \mathbb{R}^p$ to explain as much information as possible through univariate (partial) effects f_j defined on each single feature x_j and aggregated in $f_{\text{uni}}(\boldsymbol{x}) = \sum_{j=1}^{p} f_j(x_j)$. If x_j is a numerical feature, f_j is decomposed into a linear and a non-linear part, and CWB can choose to select either one or both. The linear part $b_{j,\text{lin}}$ consists of an intercept and a linear feature effect. The non-linear part $b_{j,\text{nlin}}$ describes the deviation from this linearity using a penalized B-spline effect (Eilers and Marx, 1996b) centered around the linear effect. Centering the non-linear effect allows equal degrees-of-freedom to be defined for both base learners $b_{j,\text{lin}}$ and $b_{j,\text{nlin}}$, ensuring a fair selection between these two parts and an unbiased analysis of variance. Categorical features are included as linear effects using a dummy-encoding.

The second stage builds the model part f_{pint} containing pairwise interactions f_{ij} to model the interaction between features x_i and x_j, $i \neq j$ in $\boldsymbol{x}$. We initialize the model using the predictions $\hat{f}_{\text{uni}}(\boldsymbol{x})$ from the first stage as an offset and start with the corresponding pseudo residuals. These interactions are included using bivariate interactions of categorical variables, varying coefficient models (Hastie and Tibshirani, 1993) for mixed categorical-numerical interactions, and penalized tensor-product splines (Wood, 2017) for bivariate numerical interactions. For larger p, considering all possible pairwise interactions is infeasible. We use a filtering technique to include the $\psi \cdot 100\%$ most frequently selected interactions $\mathcal{I}$ for $\psi \in (0, 1]$ in a random forest (RF) selection step (Breiman, 2001). The RF uses 500 trees and tree depth of two to allow the selection of only pairwise interactions. We then use CWB again to fit $f_{\text{pint}} = \sum_{(i,j)\in\mathcal{I}} f_{ij}(x_i, x_j)$ and refine the set of possible interactions due to its selective nature. As for the univariate model, the degrees-of-freedom are set equal for each f_{ij} to ensure a fair interaction selection. We note that the result of this stage is still interpretable, and bivariate interactions can, e.g., be visualized in a surface plot as shown in Section 4.

A final third stage is used to explain the remaining variance left after the second stage. In contrast to the previous two stages, we use here a black-box model $f_{\text{deep}}(\boldsymbol{x}) = \sum_{m=1}^{M_{\text{deep}}} T_m(\boldsymbol{x})$ including M_{deep} deep trees T_m as base learners. As for the second stage, the third stage starts with the pseudo residuals obtained after the second stage. This stage is able to represent deeper interactions and non-smooth feature effects. We can understand this stage as a measure of complexity needed to fit the given data and also as a measure of uncertainty about stages one and two. The smaller the fraction needed for stage three in the final model, the higher our confidence is with respect to the model's interpretation.

2.2 The Bigger Picture

Our framework consists of three blocks composed into an AutoML pipeline. These blocks are data preprocessing, data modelling, and HPO (illustrated in Fig. 1 as blue frames). The first preprocessing block is automatically determined by the underlying task. This includes the removal of constant features, feature encoding, and collapsing levels of high cardinal factors. It also implements an automatic imputation of missing values (see Appendix D for details). The second block contains the model fitting (Section 2.1), while the third block is formed by HPO. We use Hyperband (Li et al., 2018) here, but the framework allows using any other tuning algorithm. Common applications of Hyperband are using the number of iterations as a multifidelity budget parameter. In our three-stage approach, we determine the number of boosting iterations in each stage using early stopping. Hence, instead of the boosting iterations, a subsampling rate is used as budget parameter. More specifically, multiple candidate models are fitted on a fraction of the complete data set based on the current subsampling rate and only the most promising candidates are used further for larger fractions. Due to

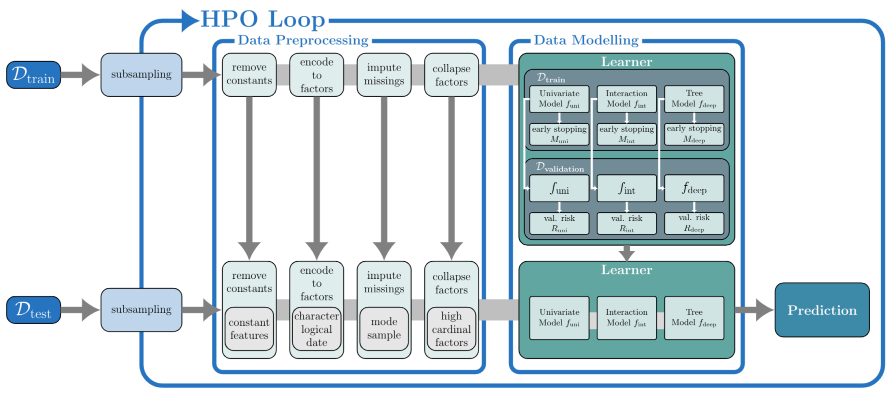
HPO Loop
Data Preprocessing
Data Modelling
$\mathcal{D}_{\text{train}}$
subsampling
remove constants
encode to factors
impute missings
collapse factors
Learner
$\mathcal{D}_{\text{train}}$
Univariate Model f_{uni}
Interaction Model f_{int}
Tree Model f_{deep}
early stopping M_{uni}
early stopping M_{int}
early stopping M_{deep}
$\mathcal{D}_{\text{validation}}$
f_{uni}
f_{int}
f_{deep}
val. risk R_{uni}
val. risk R_{int}
val. risk R_{deep}
$\mathcal{D}_{\text{test}}$
subsampling
remove constants
constant features
encode to factors
character logical date
impute missings
mode sample
collapse factors
high cardinal factors
Learner
Univariate Model f_{uni}
Interaction Model f_{int}
Tree Model f_{deep}
Prediction

Fig. 1: autocompboost's standard ML pipeline.

the choice of the fitting algorithm, another advantage is the rather small HPO search space, consisting only of HPs $\nu \in [0.001, 0.5]$ and $\psi \in [0.01, 0.2]$.

2.3 Implementation

The proposed framework is implemented in the `R` package `autocompboost` based on `compboost` (Schalk et al., 2018) and the `mlr3` ecosystem (Lang et al., 2019). Model fitting is executed in `C++`, naturally allowing for fast and parallel computing. The task-specific ML pipeline is defined via `mlr3pipelines` (Binder et al., 2021), while HPO is based on Hyperband (Li et al., 2018) using `mlr3hyperband`.

3 Benchmark

To demonstrate that the proposed framework can keep up with performances of state-of-the-art AutoML frameworks, we run our system on the OpenML AutoML Benchmark (Gijsbers et al., 2019). The benchmark is open-source – allowing for easy and fair comparisons – and includes 39 data sets. For the comparison, each AutoML system is trained for 1 hour on each outer resampling fold (10 fold cross-validation). More details can be found in Appendix E. Note that the dataset portfolio only contains classification tasks, while our framework can also be used without any restrictions for other types of tasks such as regression with different loss functions and modelling count, functional, or survival data. On a selection of 29 datasets, we run four different `autocompboost` configurations – with and without deep interactions, both with and without HPO. Additionally to the AutoML systems, Gijsbers et al. (2019) provide results for a (tuned and non-tuned) random forest and a constant predictor. We added a `glmnet` model, tuned for 1 hour via random search to the comparison. The results are shown in Figure 2 below and in detail in Table 2 in Appendix E. Results indicate that `autocompboost` is in most tasks very competitive to other existing systems.

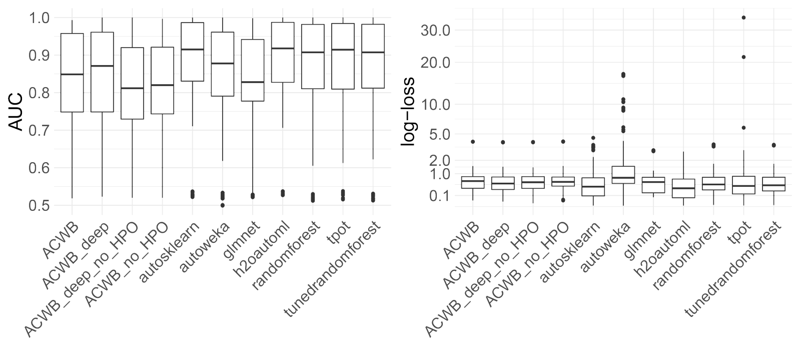

Fig. 2: Benchmark results across all datasets for different AutoML systems (Auto-WEKA, TPOT, H2O-AutoML, auto-sklearn), RF (tuned and with default values) as an ML model with good out-of-the-box performance, a GLM with elastic net regularization as a comparable interpretable model, as well as four variants of our framework (ACWB). Left: boxplots of AUC performances for binary classification tasks. Right: log-loss results for multiclass classification tasks.

4 Interpreting the framework

`autocompboost` automatically provides three important ways to interpret the final model: 1) the required complexity based on the decomposition of the model's three stages; 2) feature or variable importance (VIP); 3) the estimated partial effects f_j and f_{ij} of the first and second stage. For the sake of illustration, we use the Adult data set described in detail in Appendix G.

Required model complexity During training, the train and validation risk is logged for each stage. This allows inference about the required model complexity by calculating the percentage of explained risk per stage. The explained risk of the univariate stage can be further decomposed into risk reduction by linear and non-linear effects. In particular, the fraction of explained risk within stages one and two divided by the total explained risk describes the degree to which the interpretable model contributes to the final predictions.

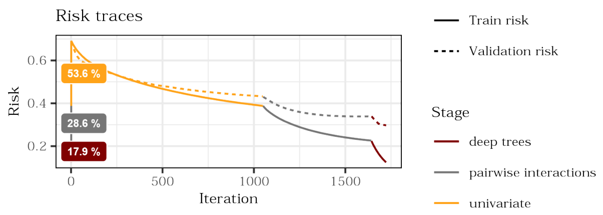

Fig. 3: Explained risk per stage and iteration mapped to the percentage of explained risk per stage as an indicator for the required complexity. Here, 53.6 % of the risk is explained by the f_{uni}, 28.6 % by f_{pint}, and 17.9 % using deep trees.

Variable importance Similar to the overall risk reduction, the VIP is the risk reduced per feature, for which we make use of the base learner structure (details are given in Appendix C). The VIP allows the user to investigate which feature in the model is able to reduce the risk and to what extent.

Explaining the model's decision-making Our routine provides two ways to explain the model's decision-making. The first visualizes the partial effects and pairwise interactions (Figure 4). The second shows how each feature contributes to the prediction score for a new observation (Figure 5 in Appendix G).

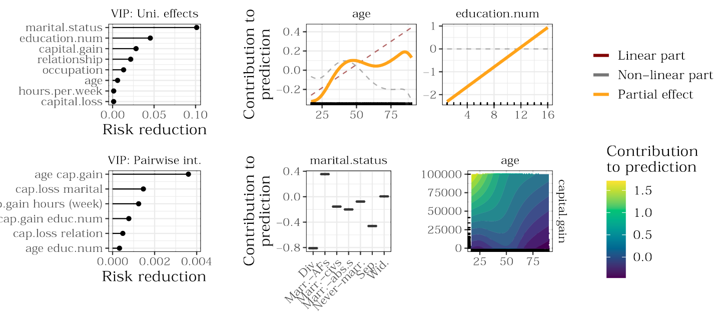

Fig. 4: VIP (left) of stages one and two and Partial effects (right) of two numerical features with their decomposition into linear and non-linear effect (right, top), a categorical feature (right, bottom left), and an interaction (right, bottom right).

Acknowledgements

This work has been partially supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A and by the Federal Ministry for Research and Technology (BMFT) grant FKZ: 01ZZ1804C (DIFUTURE, MII). The authors of this work take full responsibilities for its content.

References

Binder, M., F. Pfisterer, M. Lang, L. Schneider, L. Kotthoff, and B. Bischl (2021). mlr3pipelines - flexible machine learning pipelines in r. *Journal of Machine Learning Research 22*(184), 1–7.

Breiman, L. (2001). Random forests. *Machine learning 45*(1), 5–32.

Brockhaus, S., D. Rügamer, and S. Greven (2020). Boosting functional regression models with fdboost. *Journal of Statistical Software 94*(10), 1–50.

Bühlmann, P. and B. Yu (2003). Boosting with the L2 loss: regression and classification. *Journal of the American Statistical Association 98*(462), 324–339.

Drozdal, J., J. D. Weisz, D. Wang, G. Dass, B. Yao, C. Zhao, M. J. Muller, L. Ju, and H. Su (2020). Trust in automl: Exploring information needs for establishing trust in automated machine learning systems. *CoRR abs/2001.06509*.

Eilers, P. H. and B. D. Marx (1996a). Flexible smoothing with B-splines and penalties. *Statistical science*, 89–102.

Eilers, P. H. C. and B. D. Marx (1996b). Flexible smoothing with B-splines and penalties. *Statistical Science 11*(2), 89 – 121.

Erickson, N., J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola (2020). Autogluon-tabular: Robust and accurate automl for structured data.

Feurer, M., A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter (2019). Auto-sklearn: efficient and robust automated machine learning. In *Automated Machine Learning*, pp. 113–134. Springer, Cham.

Freitas, A. A. (2019, August). Automated machine learning for studying the trade-off between predictive accuracy and interpretability. In *Third IFIP International Cross-Domain Conference for Machine Learning and Knowledge Extraction (CD-MAKE 2019)*, Volume 11713, pp. 48–66. Springer.

Freund, Y., R. E. Schapire, et al. (1996). Experiments with a new boosting algorithm. In *icml*, Volume 96, pp. 148–156. Citeseer.

Gijsbers, P., E. LeDell, S. Poirier, J. Thomas, B. Bischl, and J. Vanschoren (2019). An open source automl benchmark. *arXiv preprint arXiv:1907.00909 [cs.LG]*. Accepted at AutoML Workshop at ICML 2019.

Hastie, T. and R. Tibshirani (1993). Varying-coefficient models. *Journal of the Royal Statistical Society: Series B (Methodological) 55*(4), 757–779.

Hofner, B., T. Hothorn, T. Kneib, and M. Schmid (2011). A framework for unbiased model selection based on boosting. *Journal of Computational and Graphical Statistics 20*(4), 956–971.

Hofner, B., A. Mayr, and M. Schmid (2016). gamboostLSS: An R package for model building and variable selection in the GAMLSS framework. *Journal of Statistical Software 74*(1).

Jamieson, K. and A. Talwalkar (2016). Non-stochastic best arm identification and hyperparameter optimization. In *Proceedings of the 19th International Con-ference on Artificial Intelligence and Statistics (AISTATS)*, pp. 240–248.

Kotthoff, L., C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown (2019). Auto-weka: Automatic model selection and hyperparameter optimization in weka. In *Automated Machine Learning*, pp. 81–95. Springer, Cham.

Lang, M., M. Binder, J. Richter, P. Schratz, F. Pfisterer, S. Coors, Q. Au, G. Casalicchio, L. Kotthoff, and B. Bischl (2019, dec). mlr3: A modern object-oriented machine learning framework in R. *Journal of Open Source Software*.

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

Li, L., K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research 18*(185), 1–52.

Liew, B. X., D. Rugamer, A. Stocker, and A. M. De Nunzio (2020). Classifying neck pain status using scalar and functional biomechanical variables – Development of a method using functional data boosting. *Gait & posture 76*, 146–150.

Meinshausen, N., G. Rocha, and B. Yu (2007). Discussion: A tale of three cousins: Lasso, L2Boosting and Dantzig. *The Annals of Statistics 35*(6), 2373 – 2384.

Pfisterer, F., J. Thomas, and B. Bischl (2019). Towards human centered automl.

R Core Team (2021). *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing.

Rügamer, D., S. Brockhaus, K. Gentsch, K. Scherer, and S. Greven (2018). Boosting factor-specific functional historical models for the detection of synchronization in bioelectrical signals. *Journal of the Royal Statistical Society: Series C (Applied Statistics) 67*(3), 621–642.

Rügamer, D. and S. Greven (2020). Inference for L2-Boosting. *Statistics and Computing 30*, 279–289.

Rügamer, D., C. Kolb, and N. Klein (2021). Semi-structured deep distributional regression: Combining structured additive models and deep learning.

Saintigny, P., L. Zhang, Y.-H. Fan, A. K. El-Naggar, V. A. Papadimitrakopoulou, L. Feng, J. J. Lee, E. S. Kim, W. K. Hong, and L. Mao (2011). Gene expression profiling predicts the development of oral cancer. *Cancer Prevention Research 4*(2), 218–229.

Schalk, D., J. Thomas, and B. Bischl (2018). compboost: Modular Framework for Component-wise Boosting. *Journal of Open Source Software 3*(30), 967.

Thomas, J., S. Coors, and B. Bischl (2018). Automatic gradient boosting. In *International Workshop on Automatic Machine Learning at ICML.*

Thomas, J., T. Hepp, A. Mayr, and B. Bischl (2017). Probing for sparse and fast variable selection with model-based boosting. *Computational and mathematical methods in medicine 2017.*

Wood, S. N. (2017). Generalized additive models: an introduction with r. Chapter 5.6, pp. 227–237. CRC press.

Xanthopoulos, I., I. Tsamardinos, V. Christophides, E. Simon, and A. Salinger (2020). Putting the human back in the automl loop. In *Proceedings of the Workshops of the EDBT/ICDT 2020 Joint Conference, Copenhagen, Denmark, March 30, 2020*, Volume 2578 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Zimmer, L., M. Lindauer, and F. Hutter (2021). Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl.

A Componentwise boosting

CWB (Bühlmann and Yu, 2003) uses gradient boosting (Freund et al., 1996) by sequentially adding one base learner b_k out of a set of base learners $\mathcal{B} = \{b_k \mid k = 1, \dots, K\}$ to the model. The objective of boosting is to minimize the empirical risk

$$\mathcal{R}_f(\mathcal{D}) = \sum_{(\boldsymbol{x},y)\in\mathcal{D}} L(y, f(\boldsymbol{x}))$$

w.r.t. a prediction model f with loss function L and data

$$\mathcal{D} = \left(\left(\boldsymbol{x}^{(1)}, y^{(1)}\right), \dots, \left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right)$$

containing the target vector $(y^{(1)}, \dots, y^{(n)})^\mathsf{T} \in \mathbb{R}^n$ and features

$$\boldsymbol{x}_j = (x_j^{(1)}, \dots, x_j^{(n)})^\mathsf{T} \in \mathbb{R}^n.$$

The collection of features is denoted as observations $\boldsymbol{x}^{(i)} = (x_1^{(i)}, \dots, x_p^{(i)})^\mathsf{T} \in \mathbb{R}^p$.

The prediction model $\hat{f}^{[m]}$ after m boosting iterations is defined using an additive structure

$$\hat{f}^{[m]}(\boldsymbol{x}) = \hat{f}^{[m-1]}(\boldsymbol{x}) + \nu b_{k^{[m]}}(\boldsymbol{x}|\hat{\boldsymbol{\theta}}^{[m]}),$$

, where the best-performing base learner $b_{k^{[m]}}(\boldsymbol{x}|\hat{\boldsymbol{\theta}}^{[m]})$ out of all base learners in $\mathcal{B}$ is added in each iteration m. To obtain an interpretable model, $b_k(\boldsymbol{x}|\boldsymbol{\theta}_k)$ is parametrized by a structured additive model with parameter $\boldsymbol{\theta}_k \in \mathbb{R}^{d_k}$ and estimate $\hat{\boldsymbol{\theta}}_k^{[m]}$ in iteration m. The offset of the model $f^{[0]} = \mathsf{arg\,min}_{c\in\mathbb{R}}\, \mathcal{R}_c(\mathcal{D})$ is found by choosing the constant c that minimizes the risk. To find the best base learner $k^{[m]} \in \{1, \dots, K\}$ in iteration m, each base learner b_k is fitted to the so-called pseudo residuals $\boldsymbol{r}^{[m]}$ with

$$r^{[m](i)} = -\frac{\partial L\left(y^{(i)}, f\left(\boldsymbol{x}^{(i)}\right)\right)}{\partial f(\boldsymbol{x}^{(i)})}\Big|_{f=\hat{f}^{[m-1]}},$$

$i = 1, \dots, n$. Therefore, the sum of squared errors

$$\mathrm{SSE}_k^{[m]}(\boldsymbol{\theta}_k) = \sum_i^n (b_k(\boldsymbol{x}^{(i)}|\boldsymbol{\theta}_k) - r^{[m](i)})^2$$

is minimized to find the best parameter estimates $\hat{\boldsymbol{\theta}}_k^{[m]} = \mathsf{arg\,min}_{\boldsymbol{\theta}_k} \mathrm{SSE}_k^{[m]}(\boldsymbol{\theta}_k)$ for all $b_k \in \mathcal{B}$. The index $k^{[m]} = \mathsf{arg\,min}_{k\in\{1,\dots,K\}} \mathrm{SSE}_k^{[m]}(\hat{\boldsymbol{\theta}}_k^{[m]})$ of the best model is then chosen by the smallest SSE. The parameter of the corresponding base learner of this iteration is defined as $\hat{\boldsymbol{\theta}}^{[m]} = \hat{\boldsymbol{\theta}}_{k^{[m]}}^{[m]}$.

A base learner b_k selects the feature(s) of the input vector $\boldsymbol{x} = (x_1, \dots, x_p)^\mathsf{T} \in \mathbb{R}^p$ required for modelling, e.g., the univariate linear regression model or splines.

This can be one feature x_i, but also two features x_i and x_j for tensor splines[2]. This allows estimating the partial feature effects as linear combination of feature (or its basis representation) and the effect $\hat{\boldsymbol{\theta}}_k$.

The iterative boosting process is performed until a predefined number of boosting iterations is reached. Another option is to use stopping mechanisms, such as early stopping, or using a time budget for a dynamic stopping of the fitting process. We denote the number of boosting iterations used to fit the model with $M \in \mathbb{N}$.

Parameter aggregation After fitting CWB, the whole model is defined by the sequence of selected base learners with estimated parameters $\{\hat{\boldsymbol{\theta}}^{[1]}, \dots, \hat{\boldsymbol{\theta}}^{[M]}\}$. We restrict the base learner to be linear in these parameters $b_j(\boldsymbol{x}|\boldsymbol{\theta}_j) = \boldsymbol{x}^\mathsf{T}\boldsymbol{\theta}_j$. Therefore, two base learners $b_{j^*}(\boldsymbol{x}|\hat{\boldsymbol{\theta}}^{[m]})$ and $b_{j^*}(\boldsymbol{x}|\hat{\boldsymbol{\theta}}^{[m']})$ of the same type j^* with parameter vectors $\hat{\boldsymbol{\theta}}^{[m]}$ and $\hat{\boldsymbol{\theta}}^{[m']}$ can be aggregated to $b_{j^*}(\boldsymbol{x}|\hat{\boldsymbol{\theta}}^{[m]} + \hat{\boldsymbol{\theta}}^{[m']})$. This is further used to obtain parameter estimates for all base learners by calculating $\hat{\boldsymbol{\theta}}_j = \nu \sum_{m=1}^{M} \sum_{k=1}^{K} \mathbb{1}_{\{j=k^{[m]}\}} \hat{\boldsymbol{\theta}}^{[m]}$ and is key for an inherent partial effects estimation and interpretation. The partial effect of the j-th feature $f_j(\boldsymbol{x})$ is defined as aggregated base learner $b_j(\boldsymbol{x}|\hat{\boldsymbol{\theta}}_j)$.

Extensions and applications CWB allows optimization of arbitrary differentiable loss functions and can thus be used for, e.g., multiclass classification, interval or survival regression, and probabilistic forecasts. It is also possible to quantify the epistemic uncertainty of CWB (Rügamer and Greven, 2020). Many other extensions of the CWB algorithm exist, such as CWB for functional data (Brockhaus et al., 2020), boosting location, scale and shape models (Hofner et al., 2016), or probing for sparse and fast variable selection (Thomas et al., 2017). Because of its interpretability properties, CWB is used in medical research, e.g., for oral cancer prediction (Saintigny et al., 2011), detection of synchronization in bioelectrical signals (Rügamer et al., 2018), or classifying pain syndromes (Liew et al., 2020).

B Univariate model

The first stage of our fitting procedure uses only univariate models, and each of the p features is included as separate base learner. Categorical features are included using a dummy encoding with a ridge penalty. Hence, if a categorical base learner is selected, all group parameters are updated at once. Numerical features are included by decomposing their effect into a linear and a non-linear effect. Hence, for each numerical feature x_j, two base learners $b_{j,\text{lin}}$ and $b_{j,\text{centered}}$ are defined. The parameter vector $\theta_{j,\text{lin}} = (\alpha_j, \beta_j)^\mathsf{T}$ of $b_{j,\text{lin}}$ contains a feature-specific intercept α_j and a slope β_j. The centered base learner $b_{j,\text{centered}}$ uses a B-spline basis (Eilers and Marx, 1996a), where the parameter vector $\gamma_j \in \mathbb{R}^{p_j}$ contains the weights of the p_j B-spline basis functions $B_1(x_j), \dots, B_{p_j}(x_j)$.

[2] It is also possible to model more complex dependencies with a base learner, e.g., when using tree base learners. In this paper with a focus on interpretability, we restrict ourselves to regression models that can be represented by linear feature effects (after a B-spline basis function evaluation).

The B-spline basis is then centered around the linear effect to subtract the linear part from the basis. The non-linear base learner is then estimated using penalized least squares with a P-spline penalty. The partial effect f_j of the numerical feature x_j is thus given as sum of the two univariate base learners $f_j = b_{j,\text{lin}} + b_{j,\text{centered}}$.

C Variable importance

The VIP of feature j is defined as

$$\text{VIP}_j = \sum_{m=1}^{M_{\text{uni}}} (\mathcal{R}_{\hat{f}_{\text{uni}}^{[m-1]}}(\mathcal{D}) - \mathcal{R}_{\hat{f}_{\text{uni}}^{[m]}}(\mathcal{D}))\mathbb{1}(k^{[m]} = j).$$

The same formula is applied to calculate the VIP_{ij} for interactions $(i, j) \in \mathcal{I}$ and $\hat{f}_{\text{pint}}$. For the Adult data, as shown in Figure 4, the two most important features during the univariate fitting stage are the marital status (`marital.status`) and a numeric representation of the education status (`education.num`), with the marital status reducing the risk twice as much as the education status. The dominating interaction is between age (`age`) and capital gain (`capital.gain`).

D The Bigger Picture

Missing factors in our framework are by default imputed by their mode, while missing numeric values are sampled from all possible values with probabilities according to their empirical distribution. Hyperband, the algorithm used for HPO, can best be understood as repeated execution of the *successive halving* procedure (Jamieson and Talwalkar, 2016).

E Benchmark setup

The original OpenML AutML benchmark consists of 39 datasets. Small- and medium-sized datasets are trained for 1 and 4 hours respectively, large datasets for 4 and 8 hours. The benchmark was initially run on Amazon Web Services *m5.2xlarge* (8 CPUs) instances inside a docker container. For our benchmark, we focus on the small to medium datasets with details described in Table 1.

We use the infrastructure of the Leibniz Supercomputing Centre (LRZ), operated by the Bavarian Academy of Sciences and Humanities. Using the `R`-package `batchtools` (Lang et al., 2017), our batch jobs ran on 8 core Haswell-based CPUs and 16Gb memory. On larger datasets and, in particular, multiclass tasks, `autocompboost` trained longer than 1 hour due to a preliminary and rather inefficient implementation using a one-versus-rest fitting procedure. Moreover, the benchmark results reveal that the implemented HPO method still offers potential for improvement.

ID	Name	# Instances	# Features	# Classes	# Missings	# Numeric Features
3	kr-vs-kp	3196	37	2	0	0
12	mfeat-factors	2000	217	10	0	216
31	credit-g	1000	21	2	0	7
53	vehicle	846	19	4	0	18
3917	kc1	2109	22	2	0	21
3945	KDDCup09_appetency	50000	231	2	8024152	192
7592	adult	48842	15	2	6465	6
9952	phoneme	5404	6	2	0	5
9977	nomao	34465	119	2	0	89
10101	blood-transfusion-service-center	748	5	2	0	4
14965	bank-marketing	45211	17	2	0	7
146195	connect-4	67557	43	3	0	0
146212	shuttle	58000	10	7	0	9
146606	higgs	98050	29	2	9	28
146818	Australian	690	15	2	0	6
146821	car	1728	7	4	0	0
146822	segment	2310	20	7	0	19
146825	Fashion-MNIST	70000	785	10	0	784
167120	numerai28.6	96320	22	2	0	21
168329	helena	65196	28	100	0	27
168330	jannis	83733	55	4	0	54
168331	volkert	58310	181	10	0	180
168335	MiniBooNE	130064	51	2	0	50
168337	guillermo	20000	4297	2	0	4296
168338	riccardo	20000	4297	2	0	4296
168908	christine	5418	1637	2	0	1599
168909	dilbert	10000	2001	5	0	2000
168911	jasmine	2984	145	2	0	8
168912	sylvine	5124	21	2	0	20

Table 1: 30 small to medium sized datasets of the OpenML AutoML Benchmark.

Metric	Dataset	ACWB	ACWB_deep	ACWB_deep_no_HPO	ACWB_no_HPO	autosklearn	autoweka	glmnet	h2oautoml	randomforest	tpot	tunedrandomforest
AUC	Australian	0.926	0.911	0.922	0.927	0.935	0.929	0.929	**0.940**	0.937	0.932	0.934
AUC	KDDCup09_appetency	0.705	0.714	0.724	0.734	0.834	0.808	0.792	**0.830**	0.786	0.824	0.786
AUC	MiniBooNE	0.963	0.965	0.958	0.955	0.985	0.961	0.936	**0.987**	0.982	0.981	0.982
AUC	adult	0.900	0.904	0.911	0.911	**0.930**	0.908	0.904	0.926	0.909	0.927	0.909
AUC	bank-marketing	0.854	0.898	0.897	0.855	**0.937**	0.827	0.909	**0.937**	0.931	0.934	0.931
AUC	blood-transfusion	0.755	0.750	0.749	0.725	**0.757**	0.741	0.754	0.756	0.686	0.724	0.689
AUC	christine	0.788	0.791	0.802	0.793	**0.830**	0.802	0.800	0.826	0.806	0.813	0.810
AUC	credit-g	0.763	0.755	0.760	0.768	0.783	0.753	0.786	0.789	0.795	0.786	**0.796**
AUC	guillermo	0.786	0.780	0.752	0.753	0.901	0.878	0.771	**0.910**	0.903	0.819	0.903
AUC	higgs	0.745	0.746	0.761	0.762	0.793	0.677	0.680	**0.814**	0.803	0.802	0.803
AUC	jasmine	0.841	0.848	0.849	0.843	0.884	0.861	0.849	0.888	0.888	0.885	**0.889**
AUC	kc1	0.790	0.791	0.793	0.795	0.840	0.814	0.799	0.836	0.836	0.841	**0.842**
AUC	kr-vs-kp	0.976	0.999	0.998	0.994	**1.000**	0.976	0.995	**1.000**	0.999	**1.000**	**1.000**
AUC	nomao	0.985	0.984	0.988	0.989	**0.996**	0.984	0.988	**0.996**	0.995	0.995	0.995
AUC	numerai28.6	0.528	0.528	0.527	0.526	0.529	0.520	0.529	**0.532**	0.520	0.525	0.521
AUC	phoneme	0.906	0.909	0.920	0.919	0.963	0.957	0.813	0.968	0.965	**0.969**	0.966
AUC	riccardo	0.949	0.956	0.756	0.847	**1.000**		0.996	**1.000**	0.999	0.992	**1.000**
AUC	sylvine	0.972	0.973	0.977	0.974	0.990	0.975	0.966	0.990	0.983	**0.992**	0.984
log-loss	Fashion-MNIST	0.608	0.589	0.613	0.615	0.354	0.581	0.407	**0.294**	0.361	0.651	0.362
log-loss	car	0.553	0.200	0.312	0.427	0.010	0.243	0.161	0.002	0.144	**0.000**	0.047
log-loss	connect-4	0.837	0.787	0.812	0.793	0.426	0.741	0.606	**0.345**	0.495	0.400	0.478
log-loss	dilbert	0.377	0.283	0.424	0.433	0.097	584.504	0.146	**0.052**	0.328	0.217	0.329
log-loss	helena	3.977	3.921	3.925	3.998	3.447	14.102	2.922	**2.800**	3.550	3.245	3.559
log-loss	jannis	0.870	0.871	0.854	0.857	0.705	1077.099	0.832	**0.681**	0.728	0.732	0.729
log-loss	mfeat-factors	0.267	0.245	0.152	0.173	**0.099**	0.627	0.106	0.105	0.234	0.138	0.201
log-loss	segment	0.392	0.307	0.232	0.273	0.060	0.501	0.325	**0.047**	0.084	0.052	0.069
log-loss	shuttle	0.087	0.028	0.024	0.059	0.001	0.015	0.118	**0.000**	0.001	3.444	0.001
log-loss	vehicle	0.637	0.583	0.546	0.546	0.395	2.105	0.423	**0.353**	0.497	0.414	0.486
log-loss	volkert	1.499	1.476	1.411	1.471	0.945	1.110	1.177	**0.821**	0.980	1.011	0.979

Table 2: Results of the benchmark experiment based on 29 datasets (see Table 1). Results are presented as mean scores of a 10-fold crossvalidation. Best scores are presented as bold numbers. For the binary and multiclass classification tasks, the results are given by the AUC and the log-loss, respectively.

F Required model complexity

During model training of `autocompboost` for the three described stages, we denote the number of boosting iterations of each stage with M_{uni}, M_{pint}, and M_{deep}. Furthermore, the empirical risk of the intercept model $\mathcal{R}_0 = \mathcal{R}_{\hat{f}^{[0]}}(\mathcal{D})$, the univariate model $\mathcal{R}_{\text{uni}} = \mathcal{R}_{\hat{f}_{\text{uni}}^{[M_{\text{uni}}]}}(\mathcal{D})$, the pairwise interaction model $\mathcal{R}_{\text{pint}} = \mathcal{R}_{\hat{f}_{\text{pint}}^{[M_{\text{pint}}]}}(\mathcal{D})$, and the deep interaction model $\mathcal{R}_{\text{deep}} = \mathcal{R}_{\hat{f}_{\text{deep}}^{[M_{\text{deep}}]}}(\mathcal{D})$ is logged. We define the fraction of explained risk per stage with

$$\begin{aligned}
\rho_{\text{uni}} &= (\mathcal{R}_0 - \mathcal{R}_{\text{uni}})/\delta_{\mathcal{R}} \\
\rho_{\text{pint}} &= (\mathcal{R}_{int} - \mathcal{R}_{\text{pint}})/\delta_{\mathcal{R}} \\
\rho_{\text{deep}} &= (\mathcal{R}_{pint} - \mathcal{R}_{\text{deep}})/\delta_{\mathcal{R}}
\end{aligned}$$

with $\delta_{\mathcal{R}} = \mathcal{R}_0 - \mathcal{R}_{\text{deep}}$. The value of ρ indicates how much of the overall explained risk is explained by stage the respective stage. Hence, ρ is an indicator of how much complexity of the model is required to obtain different levels of prediction accuracy.

G Application

Adult data set The Adult data was collected by the Census bureau. The binary classification task is to predict whether an adult earns more than \$50,000. The given features are, for example, education, hours of work per week, or marital status. The predicted scores $\hat{f} \in \mathbb{R}$ are mapped via the logistic function $s(\hat{f}) = (1 + \exp(-\hat{f}))^{-1} \in (0, 1)$ to $(0, 1)$, which can be interpreted as probabilities for the positive class of earning more than \$50,000. Hence, a predicted partial effect greater than zero favors the prediction of the positive class.

Required model complexity Figure 3 demonstrates that, for the Adult data, 53.6 % of the risk is already explained by the univariate model and 28.6 % by the pairwise interactions. Including deep trees into the model accounts for another 17.9 % explained risk.

Explaining the models decision making Figure 4 (middle) shows how partial effects of both numerical and categorical features can be visualized. In our example, the effect of the feature `age` shows both linear and non-linear effects. The second most important feature `education.num` only selects the linear component.

Pairwise interactions f_{ij} are visualized by plotting the effect surface in the two feature dimensions. Figure 4 demonstrates this, showing that especially younger adults with large capital gain likely have earnings greater than \$50,000 per year.

If the third stage has a significant contribution for explaining the final model fit, interpretation based on the first two stages must be performed with caution due to two reasons. First, the major part of the model's prediction stems from the black box model, and interpreting the structured partial effects alone is likely to be misleading. Second, a potential overlap in the hypothesis space between the interpretable stages and the black box stage might yield to an identifiability issue (see, e.g., Rügamer et al., 2021).

Prediction decomposition `autocompboost` also allows the user to better understand the system's decision-making process when a new observation $\boldsymbol{x}_0$ is used to predict the score $\hat{f}(\tilde{\boldsymbol{x}})$. In this case, it can visualize the contribution of all univariate effects by calculating $f_j(\tilde{x}_j)$, pairwise interactions with $f_{ij}(\tilde{x}_i, \tilde{x}_j)$, and the black box part $f_{\text{deep}}(\tilde{\boldsymbol{x}})$. Figure 5 shows this decomposition. The contribution of the black box part is summarized as "deep trees" contribution.

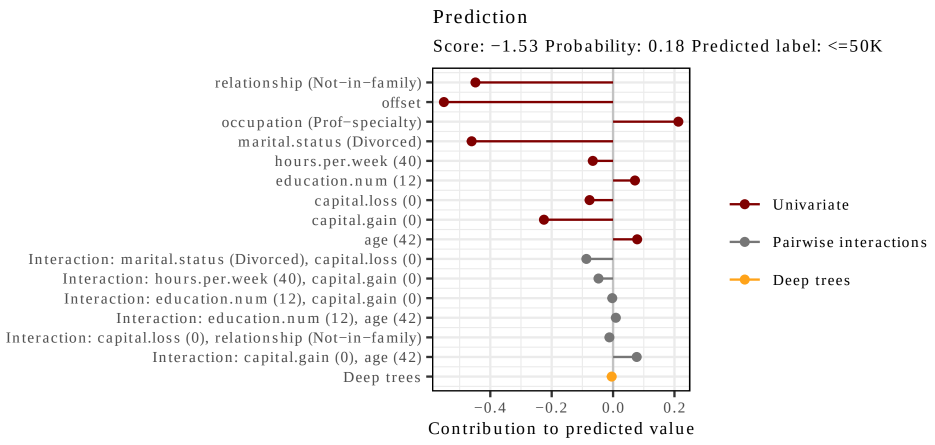

Fig. 5: Decomposition of how a new predicted score is calculated.

User-friendly interface To further simplify the use of `autocompboost`, the previous explained techniques can be interactively visualized in a dashboard. Thereby, the user automatically obtains a pre-selection of important features, effects, and corresponding visualizations by only a few clicks. It is also possible to move the whole `autocompboost` pipeline into the dashboard, i.e., perform 1) task-creation, 2) data modelling and 3) interpretation in order to make interpretable ML models available to experienced ML users as well as a larger group of non-ML experts.

CHAPTER 9

Privacy-Preserving and Lossless Distributed Estimation of High-Dimensional Generalized Additive Mixed Models

Contributing article

Schalk, D., Bischl, B., and Rügamer, D. (2023a). Privacy-preserving and lossless distributed estimation of high-dimensional generalized additive mixed models. *arXiv preprint arXiv:2210.07723*

At the time the thesis was handed in, the article was in review at the journal *Statistics and Computing*.

Declaration of contributions

Daniel Schalk developed the methodology, constructed the theory, and implemented it in an R package[1]. With this package, he created the application example and the comparison to state-of-the-art approaches as well as all graphics and tables. The publication and interpretation of the results were mainly written by him, except for specific sections (see the contribution of the co-authors).

Contribution of the coauthors

David Rügamer worked out the connection to GAMMs, wrote the corresponding text passages, and assisted in setting the algorithm parameters for comparison with state-of-the-art approaches and interpreting the results. Bernd Bischl and David Rügamer helped with revising the manuscript.

[1]https://github.com/schalkdaniel/dsCWB

Privacy-Preserving and Lossless Distributed Estimation of High-Dimensional Generalized Additive Mixed Models

Schalk Daniel[1,2*], Bischl Bernd[1,2] and Rügamer David[1,2,3]

[1*]Department of Statistics, LMU Munich, Munich, Germany.
[2]Munich Center for Machine Learning (MCML).
[3]Department of Statistics, TU Dortmund, Dortmung, Germany.

*Corresponding author(s). E-mail(s): daniel.schalk@stat.uni-muenchen.de;
Contributing authors: bernd.bischl@stat.uni-muenchen.de;
david.ruegamer@stat.uni-muenchen.de;

Abstract

Various privacy-preserving frameworks that respect the individual's privacy in the analysis of data have been developed in recent years. However, available model classes such as simple statistics or generalized linear models lack the flexibility required for a good approximation of the underlying data-generating process in practice. In this paper, we propose an algorithm for a distributed, privacy-preserving, and lossless estimation of generalized additive mixed models (GAMM) using component-wise gradient boosting (CWB). Making use of CWB allows us to reframe the GAMM estimation as a distributed fitting of base learners using the $\boldsymbol{L_2}$-loss. In order to account for the heterogeneity of different data location sites, we propose a distributed version of a row-wise tensor product that allows the computation of site-specific (smooth) effects. Our adaption of CWB preserves all the important properties of the original algorithm, such as an unbiased feature selection and the feasibility to fit models in high-dimensional feature spaces, and yields equivalent model estimates as CWB on pooled data. Next to a derivation of the equivalence of both algorithms, we also showcase the efficacy of our algorithm on a distributed heart disease data set and compare it with state-of-the-art methods.

Keywords: Distributed Computing, Functional Gradient Descent, Generalized Linear Mixed Model, Machine Learning, Privacy-preserving Modelling

1 Introduction

More than ever, data is collected to record the ubiquitous information in our everyday life. However, on many occasions, the physical location of data points is not confined to one place (one global site) but distributed over different locations (sites). This is the case for, e.g., patient records that are gathered at different hospitals but usually not shared between hospitals or other facilities due to the sensitive information they contain. This makes data analysis challenging, particularly if methods require or notably benefit from incorporating all available (but distributed) information. For example, personal patient information is typically distributed over several hospitals, while sharing or merging different data sets in a central location is prohibited. To overcome this limitation, different approaches have been developed to directly operate at different sites and unite information without having to share sensitive parts of the data to allow privacy-preserving data analysis.

Distributed Data

Distributed data can be partitioned vertically or horizontally across different sites. Horizontally partitioned data means that observations are spread across different sites with access to all existing features of the available data point, while for vertically partitioned data, different sites have access to all observations but different features (covariates) for each of these observations. In this work, we focus on horizontally partitioned data. Existing approaches for horizontally partitioned data vary from fitting regression models such as generalized linear models (GLMs; Wu et al, 2012; Lu et al, 2015; Jones et al, 2013; Chen et al, 2018), to conducting distributed evaluations (Boyd et al, 2015; Ünal et al, 2021; Schalk et al, 2022b), to fitting artificial neural networks (McMahan et al, 2017). Furthermore, various software frameworks are available to run a comprehensive analysis of distributed data. One example is the collection of `R` (R Core Team, 2021) packages `DataSHIELD` (Gaye et al, 2014), which enables data management and descriptive data analysis as well as securely fitting of simple statistical models in a distributed setup without leaking information from one site to the others.

Interpretability and Data Heterogeneity

In many research areas that involve critical decision-making, especially in medicine, methods should not only excel in predictive performance but also be interpretable. Models should provide information about the decision-making process, the feature effects, and the feature importance as well as intrinsically select important features. Generalized additive models (GAMs; see, e.g., Wood, 2017) are one of the most flexible approaches in this respect, providing an interpretable yet complex models that also allow for non-linearity in the data.

As longitudinal studies are often the most practical way to gather information in many research fields, methods should also be able to account for subject-specific effects and account for the correlation of repeated measurements. Furthermore, when analyzing data originating from different sites, the assumption of having identically distributed observations across all sites often does not hold. In this case, a reasonable assumption for the data-generating process is a site-specific deviation from the general population mean. Adjusting models to this situation is called interoperability (Litwin et al, 1990), while ignoring it may lead to biased or wrong predictions.

1.1 Related Literature

Various approaches for distributed and privacy-preserving analysis have been proposed in recent years. In the context of statistical models, Karr et al (2005) describe how to calculate a linear model (LM) in a distributed and privacy-preserving fashion by sharing data summaries. Jones et al (2013) propose a similar approach for GLMs by communicating the Fisher information and score vector to conduct a distributed Fisher scoring algorithm. The site information is then globally aggregated to estimate the model parameters. Other privacy-preserving techniques include ridge regression (Chen et al, 2018), logistic regression, and neural networks (Mohassel and Zhang, 2017).

In machine learning, methods such as the naive Bayes classifier, trees, support vector machines, and random forests (Li et al, 2020a) exist with specific encryption techniques (e.g., the Paillier cryptosystem; Paillier, 1999) to conduct model updates. In these setups, a trusted third party is usually required. However, this is often unrealistic and difficult to implement, especially in a medical or clinical setup. Furthermore, as encryption is an expensive operation, its application is infeasible for complex algorithms that require many encryption calls (Naehrig et al, 2011). Existing privacy-preserving boosting techniques often focus on the AdaBoost algorithm by using aggregation techniques of the base classifier (Lazarevic and Obradovic, 2001; Gambs et al, 2007). A different approach to boosting decision trees in a federated learning setup was introduced by (Li et al, 2020b) using a locality-sensitive hashing to obtain similarities between data sets without sharing private information. These algorithms focus on aggregating tree-based base components, making them difficult to interpret, and come with no inferential guarantees.

In order to account for repeated measurements, Luo et al (2022) propose a privacy-preserving and lossless way to fit linear mixed models (LMMs) to correct for heterogeneous site-specific random effects. Their concept of only sharing aggregated values is similar to our approach, but is limited in the complexity of the model and only allows normally distributed outcomes. Other methods to estimate LMMs in a secure and distributed fashion are Zhu et al (2020), Anjum et al (2022), or Yan et al (2022).

Besides privacy-preserving and distributed approaches, integrative analysis is another technique based on pooling the data sets into one and analyzing this pooled data set while considering challenges such as heterogeneity or the curse of dimensionality (Curran and Hussong, 2009; Bazeley, 2012; Mirza et al, 2019). While advanced from a technical perspective by, e.g., outsourcing computational demanding tasks such as the analysis of multi-omics data to cloud services (Augustyn et al, 2021), the existing statistical cloud-based methods only deal with basic statistics. The challenges of integrative analysis are similar to the ones tackled in this work, our approach, however, does not allow merging the data sets in order to preserve privacy.

1.2 Our Contribution

This work presents a method to fit generalized additive mixed models (GAMMs) in a privacy-preserving and lossless manner[1] to horizontally distributed data. This not only allows the incorporation of site-specific random effects and accounts for repeated measurements in LMMs, but also facilitates the estimation of mixed models with responses following any distribution from the exponential family and provides the possibility to estimate complex non-linear relationships between covariates and the response. To the best of our knowledge, we are the first to provide an algorithm to fit the class of GAMMs in a privacy-preserving and lossless fashion on distributed data.

Our approach is based on component-wise gradient boosting (CWB; Bühlmann and Yu, 2003). CWB can be used to estimate additive models, account for repeated measurements, compute feature importance, and conduct feature selection. Furthermore, CWB is suited for high-dimensional data situations ($n \ll p$). CWB is therefore often used in practice for, e.g., predicting the development of oral cancer (Saintigny et al, 2011), classifying individuals with and without patellofemoral pain syndrome (Liew et al, 2020), or detecting synchronization in bioelectrical signals (Rügamer et al, 2018). However, there have so far not been any attempts to allow for a distributed, privacy-preserving, and lossless computation of the CWB algorithm. In this paper, we propose a distributed version of CWB that yields the identical model produced by the original algorithm on pooled data and that accounts for site heterogeneity by including interactions between features and a site variable. This is achieved by adjusting the fitting process using 1) a distributed estimation procedure, 2) a distributed version of row-wise tensor product base learners, and 3) an adaption of the algorithm to conduct feature selection in the distributed setup. We implement our method in `R` using the `DataSHIELD` framework and demonstrate its application in an exemplary medical data analysis. Our distributed version of the original CWB algorithm does not have any additional HPs and uses optimization strategies from previous research results to define meaningful values for all hyperparameters, effectively yielding a tuning-free method.

The remainder of this paper is structured as follows: First, we introduce the basic notation, terminology, and setup of GAMMs in Section 2. We then describe the original CWB algorithm in Section 2.3 and its link to GAMMs. In Section 3, we present the distributed setup and our novel extension of the CWB algorithm. Finally, Section 4 demonstrates both how our distributed CWB algorithm can be used in practice and how to interpret the obtained results.

Implementation

We implement our approach as an `R` package using the DataSHIELD framework and make it available on GitHub[2]. The code for the analysis can also be found in the repository[3].

[1] In this article, we define a distributed fitting procedure as lossless if the model parameters of the algorithm are the same as the ones computed on the pooled data.

[2] github.com/schalkdaniel/dsCWB

[3] github.com/schalkdaniel/dsCWB/blob/main/usecase/analyse.R

2 Background

2.1 Notation and Terminology

Our proposed approach uses the CWB algorithm as fitting engine. Since this method was initially developed in machine learning, we introduce here both the statistical notation used for GAMMs as well as the respective machine learning terminology and explain how to relate the two concepts.

We assume a p-dimensional covariate or feature space $\mathcal{X} = (\mathcal{X}_1 \times \ldots \times \mathcal{X}_p) \subseteq \mathbb{R}^p$ and response or outcome values from a target space $\mathcal{Y}$. The goal of boosting is to find the unknown relationship f between $\mathcal{X}$ and $\mathcal{Y}$. In turn, GAMMs (as presented in Section 2.2) model the conditional distribution of an outcome variable Y with realizations $y \in \mathcal{Y}$, given features $\boldsymbol{x} = (x_1, \ldots, x_p) \in \mathcal{X}$. Given a data set $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(1)}, y^{(1)}\right), \ldots, \left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}$ with n observations drawn (conditionally) independently from an unknown probability distribution $\mathbb{P}_{xy}$ on the joint space $\mathcal{X} \times \mathcal{Y}$, we aim to estimate this functional relationship in CWB with $\hat{f}$. The goodness-of-fit of a given model $\hat{f}$ is assessed by calculating the empirical risk $\mathcal{R}_\text{emp}(\hat{f}) = n^{-1} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} L(y, \hat{f}(\boldsymbol{x}))$ based on a loss function $L : \mathcal{Y} \times \mathbb{R} \to \mathbb{R}$ and the data set $\mathcal{D}$. Minimizing $\mathcal{R}_\text{emp}$ using this loss function is equivalent to estimating f using maximum likelihood by defining $L(y, f(\boldsymbol{x})) = -\ell(y, h(f(\boldsymbol{x})))$ with log-likelihood ℓ, response function h and minimizing the sum of log-likelihood contributions.

In the following, we also require the vector $\boldsymbol{x}_j = (x_j^{(1)}, \ldots, x_j^{(n)})^\top \in \mathcal{X}_j$, which refers to the j^th feature. Furthermore, let $\boldsymbol{x} = (x_1, \ldots, x_p)$ and y denote arbitrary members of $\mathcal{X}$ and $\mathcal{Y}$, respectively. A special role is further given to a subset $\boldsymbol{u} = (u_1, \ldots, u_q)^\top$, $q \leq p$, of features $\boldsymbol{x}$, which will be used to model the heterogeneity in the data.

2.2 Generalized Additive Mixed Models

A very flexible class of regression models to model the relationship between covariates and the response are GAMMs (see, e.g., Wood, 2017). In GAMMs, the response $Y^{(i)}$ for observation $i = 1, \ldots, n_s$ of measurement unit (or site) s is assumed to follow some exponential family distribution such as the Poisson, binomial, or normal distributions (see, e.g., McCullagh and Nelder, 2019), conditional on features $\boldsymbol{x}^{(i)}$ and the realization of some random effects. The expectation $\mu := \mathbb{E}(Y^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)})$ of the response $Y^{(i)}$ for observations $i = 1, \ldots, n_s$ of measurement unit (or site) s in GAMMs is given by

$$\begin{aligned} h^{-1}(\mu^{(i)}) &= f^{(i)} \\ &= \sum_{j\in\mathcal{J}_1} x_j^{(i)}\beta_j + \sum_{j\in\mathcal{J}_2} u_j^{(i)}\gamma_{j,s} + \sum_{j\in\mathcal{J}_3} \phi_j(x_j^{(i)}). \end{aligned} \tag{1}$$

In (1), h is a smooth monotonic response function, f corresponds to the additive predictor, $\gamma_{j,s} \sim \mathcal{N}(0, \psi)$ are random effects accounting for heterogeneity in the data, and ϕ_j are non-linear effects of pre-specified covariates. The different index sets $\mathcal{J}_1, \mathcal{J}_2, \mathcal{J}_3 \subseteq \{1, \ldots, p\} \cup \emptyset$ indicate which features are modeled as fixed effects, random effects, or non-linear (smooth) effects, respectively. The modeler usually defines these sets. However, as we will also explain later, the use of CWB as a fitting engine allows for automatic feature selection and therefore does not require explicitly defining these sets. In GAMMs, smooth effects are usually represented by (spline) basis functions, i.e., $\phi_j(x_j) \approx (B_{j,1}(x_j), \ldots, B_{j,d_j}(x_j))^\top \boldsymbol{\theta}_j$, where $\boldsymbol{\theta}_j \in \mathbb{R}^{d_j}$ are the basis coefficients corresponding to each basis function B_{j,d_j}. The coefficients are typically constrained in their flexibility by adding a quadratic (difference) penalty for (neighboring) coefficients to the objective function to enforce smoothness. GAMMs, as in (1), are not limited to univariate smooth effects ϕ_j, but allow for higher-dimensional non-linear effects $\phi(x_{j_1}, x_{j_2}, \ldots, x_{j_k})$. The most common higher-dimensional smooth interaction effects are bivariate effects ($k = 2$) and can be represented using a bivariate or a tensor product spline basis (see Section 2.3.1 for more details). Although higher-order splines with $k > 2$ are possible, models are often restricted to bivariate interactions for the sake of interpretability and computational feasibility. In Section 3, we will further introduce varying coefficient terms $\phi_{j,s}(x_j)$ in the model (1), i.e., smooth effects f varying with a second variable s. Analogous to random slopes, s can also be the index set defining observation units of random effects $\mathcal{J}_2$. Using an appropriate distribution assumption for the basis coefficients $\boldsymbol{\theta}_j$, these varying coefficients can then be considered as random smooth effects.

2.3 Component-Wise Boosting

Component-wise (gradient) boosting (CWB; Bühlmann and Yu, 2003; Bühlmann et al, 2007) is a an iterative algorithm that performs block-coordinate descent steps with blocks (or base learners) corresponding to the additive terms in (1). With a suitable choice of base learners and objective function, CWB allows efficient optimization of GAMMs, even in high-dimensional settings with $p \gg n$. We will first introduce the concept of base learners that embed additive terms of the GAMM into boosting and subsequently describe the actual fitting routine of CWB. Lastly, we will describe the properties of the algorithm and explain its connection to model (1).

2.3.1 Base Learners

In CWB, the l^{th} base learner $b_l : \mathcal{X} \to \mathbb{R}$ is used to model the contribution of one or multiple features in the model. In this work, we investigate parametrized base learners $b_l(\boldsymbol{x}, \boldsymbol{\theta}_l)$ with parameters $\boldsymbol{\theta}_l \in \mathbb{R}^{d_l}$. For simplicity, we will use $\boldsymbol{\theta}$ as a wildcard for the coefficients of either fixed effects, random effects, or spline bases in the following. We assume that each base learner can be represented by a generic basis representation $g_l : \mathcal{X} \to \mathbb{R}^{d_l}$, $\boldsymbol{x} \mapsto g_l(\boldsymbol{x}) = (g_{l,1}(\boldsymbol{x}), \dots, g_{l,d_l}(\boldsymbol{x}))^\mathsf{T}$ and is linear in the parameters, i.e., $b_l(\boldsymbol{x}, \boldsymbol{\theta}_l) = g_l(\boldsymbol{x})^\mathsf{T}\boldsymbol{\theta}_l$. For n observations, we define the design matrix of a base learner b_l as $\boldsymbol{Z}_l := (g_l(\boldsymbol{x}^{(1)}), \dots, g_l(\boldsymbol{x}^{(n)}))^\mathsf{T} \in \mathbb{R}^{n \times d_l}$. Note that base learners are typically not defined on the whole feature space but on a subset $\mathcal{X}_l \subseteq \mathcal{X}$. For example, a common choice for CWB is to define one base learner for every feature $x_l \in \mathcal{X}_l$ to model the univariate contributions of that feature.

A base learner $b_l(\boldsymbol{x}, \boldsymbol{\theta}_l)$ can depend on HPs $\boldsymbol{\alpha}_l$ that are set prior to the fitting process. For example, choosing a base learner using a P-spline (Eilers and Marx, 1996) representation requires setting the degree of the basis functions, the order of the difference penalty term, and a parameter λ_l determining the smoothness of the spline. In order to represent GAMMs in CWB, the following four base learner types are used.

(Regularized) linear base learners

A linear base learner is used to include linear effects of a features $x_{j_1}, \dots, x_{j_{d_l}}$ into the model. The basis transformation is given by $g_l(\boldsymbol{x}) = (g_{l,1}(\boldsymbol{x}), \dots, g_{l,d_l+1}(\boldsymbol{x}))^\mathsf{T} = (1, x_{j_1}, \dots, x_{j_{d_l}})^\mathsf{T}$. Linear base learners can be regularized by incorporating a ridge penalization (Hoerl and Kennard, 1970) with tunable penalty parameter λ_l as an HP $\boldsymbol{\alpha}_l$. Fitting a ridge penalized linear base learner to a response vector $\boldsymbol{y} \in \mathbb{R}^n$ results in the penalized least squares estimator $\hat{\boldsymbol{\theta}}_l = (\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{Z}_l + \boldsymbol{K}_l)^{-1}\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{y}$ with penalty matrix $\boldsymbol{K}_l = \lambda_l \boldsymbol{I}_{d_l+1}$, where $\boldsymbol{I}_d$ denotes the d-dimensional identity matrix. Often, an unregularized linear base learner is also included to model the contribution of one feature $\boldsymbol{x}_j$ as a linear base learner without penalization. The basis transformation is then given by $g_l(\boldsymbol{x}) = (1, x_j)^\mathsf{T}$ and $\lambda_l = 0$.

Spline base learners

These base learners model smooth effects using univariate splines. A common choice is penalized B-splines (P-Splines; Eilers and Marx, 1996), where the feature x_j is transformed using a B-spline basis transformation $g_l(\boldsymbol{x}) = (B_{l,1}(x_j), \dots, B_{l,d_l}(x_j))^\mathsf{T}$ with d_l basis functions $g_{l,m} = B_{l,m}$, $m = 1, \dots, d_l$. In this case, the choice of the spline order B, the number of basis functions d_l, the penalization term λ_l, and the order v of the difference penalty (represented by a matrix $\boldsymbol{D}_l \in \mathbb{R}^{d_{l-v} \times d_l}$) are considered HPs $\boldsymbol{\alpha}_l$ of the base learner. The base learner's parameter estimator in general is given by the penalized least squares solution $\hat{\boldsymbol{\theta}}_l = (\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{Z}_l + \boldsymbol{K}_l)^{-1}\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{y}$, with penalization matrix $\boldsymbol{K}_l = \lambda_l \boldsymbol{D}_l^\mathsf{T}\boldsymbol{D}_l$ in the case of P-splines.

Categorical and random effect base learners

Categorical features $x_j \in \{1, \dots, G\}$ with $G \in \mathbb{N}, G \geq 2$ classes are handled by a binary encoding $g_l(\boldsymbol{x}) = (\mathbb{1}_{\{1\}}(x_j), \dots, \mathbb{1}_{\{G\}}(x_j))^\mathsf{T}$ with the indicator function $\mathbb{1}_A(x) = 1$ if $x \in A$ and $\mathbb{1}_A(x) = 0$ if $x \notin A$. A possible alternative encoding is the dummy encoding with $\breve{g}_l(\boldsymbol{x}) = (1, \mathbb{1}_{\{1\}}(x_j), \dots, \mathbb{1}_{\{G-1\}}(x_j))^\mathsf{T}$ with reference group G. Similar to linear and spline base learners, it is possible incorporate a ridge penalization with HP $\boldsymbol{\alpha}_l = \lambda_l$. This results in the base learner's penalized least squared estimator

$\hat{\boldsymbol{\theta}}_l = (\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{Z}_l + \boldsymbol{K}_l)^{-1}\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{y}$ with penalization matrix $\boldsymbol{K}_l = \lambda_l \boldsymbol{I}_G$. Due to the mathematical equivalence of ridge penalized linear effects and random effects with normal prior (see, e.g., Brumback et al, 1999), this base learner can further be used to estimate random effect predictions $\hat{\gamma}_j$ when using categorical features u_j and thereby account for heterogeneity in the data.

Row-wise tensor product base learners

This type of base learner is used to model a pairwise interaction between two features x_j and x_k. Given two base learners b_j and b_k with basis representations $g_j(\boldsymbol{x}) = (g_{j,1}(x_j), \dots, g_{j,d_j}(x_j))^\mathsf{T}$ and $g_k(\boldsymbol{x}) = (g_{k,1}(x_k), \dots, g_{k,d_k}(x_k))^\mathsf{T}$, the basis representation of the row-wise tensor product base learner $b_l = b_j \times b_k$ is defined as $g_l(\boldsymbol{x}) = (g_j(\boldsymbol{x})^\mathsf{T} \otimes g_k(\boldsymbol{x})^\mathsf{T})^\mathsf{T} = (g_{j,1}(x_j)g_k(\boldsymbol{x})^\mathsf{T}, \dots, g_{j,d_j}(x_j)g_k(\boldsymbol{x})^\mathsf{T})^\mathsf{T} \in \mathbb{R}^{d_l}$ with $d_l = d_j d_k$. The HPs $\boldsymbol{\alpha}_l = \{\boldsymbol{\alpha}_j, \boldsymbol{\alpha}_k\}$ of a row-wise tensor product base learner are induced by the HPs $\boldsymbol{\alpha}_j$ and $\boldsymbol{\alpha}_k$ of the respective individual base learners. Analogously to other base learners, the penalized least squared estimator in this case is $\hat{\boldsymbol{\theta}}_l = (\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{Z}_l + \boldsymbol{K}_l)^{-1}\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{y}$ with penalization matrix $\boldsymbol{K}_l = \tau_j \boldsymbol{K}_j \otimes \boldsymbol{I}_{d_k} + \boldsymbol{I}_{d_j} \otimes \tau_k \boldsymbol{K}_k \in \mathbb{R}^{d_l \times d_l}$. This Kronecker sum penalty, in particular, allows for anisotropic smoothing with penalties τ_j and τ_k when using two spline bases for g_j and g_k, and varying coefficients or random splines when combining a (penalized) categorical base learner and a spline base learner.

2.3.2 Fitting Algorithm

CWB first initializes an estimate $\hat{f}$ of the additive predictor with a loss-optimal constant value $\hat{f}^{[0]} = \arg\min_{c\in\mathbb{R}} \mathcal{R}_\text{emp}(c)$. It then proceeds and estimates Eq. (1) using an iterative steepest descent minimization in function space by fitting the previously defined base learners to the model's functional gradient $\nabla_f L(y, f)$ evaluated at the current model estimate $\hat{f}$. Let $\hat{f}^{[m]}$ denote the model estimation after $m \in \mathbb{N}$ iterations. In each step in CWB, the pseudo residuals $\tilde{r}^{[m](i)} = -\nabla_f L(y^{(i)}, f(\boldsymbol{x}^{(i)}))|_{f=\hat{f}^{[m-1]}}$ for $i \in \{1, \dots, n\}$ are first computed. CWB then selects the best-fitting base learner from a pre-defined pool of base-learners denoted by $\mathcal{B} = \{b_l\}_{l\in\{1,\dots,|\mathcal{B}|\}}$ and adds the base learner's contribution to the previous model $\hat{f}^{[m]}$. The selected base learner is chosen based on its sum of squared errors (SSE) when regressing the pseudo residuals $\tilde{\boldsymbol{r}}^{[m]} = (\boldsymbol{r}^{[m](1)}, \dots, \boldsymbol{r}^{[m](n)})^\mathsf{T}$ onto the base learner's features using the L_2-loss. Further details of CWB are given in Algorithm 1 (see, e.g., Schalk et al, 2022a).

Controling HPs of CWB

Good estimation performance can be achieved by selecting a sufficiently small learning rate, e.g., 0.01, as suggested in Bühlmann et al (2007), and adaptively selecting the number of boosting iterations via early stopping on a validation set. To enforce a fair selection of model terms and thus unbiased effect estimation, regularization parameters are set such that all base learners have the same degrees-of-freedom (Hofner et al, 2011). As noted by Bühlmann et al (2007), choosing smaller degrees-of-freedom induces more penalization (and thus, e.g., smoother estimated function for spline base learners), which yields a model with lower variance at the cost of a larger bias. This bias induces a shrinkage in the estimated coefficients towards zero but can be reduced by running the optimization process for additional iterations.

Algorithm 1 Vanilla CWB algorithm

Input Train data $\mathcal{D}$, learning rate ν, number of boosting iterations M, loss function L, set of base learner $\mathcal{B}$
Output Model $\hat{f}^{[M]}$ defined by fitted parameters $\hat{\boldsymbol{\theta}}^{[1]}, \dots, \hat{\boldsymbol{\theta}}^{[M]}$

```
1: procedure CWB($\mathcal{D}, \nu, L, \mathcal{B}$)
2:     Initialize: $\hat{f}^{[0]}(\boldsymbol{x}) = \arg\min_{c\in\mathbb{R}} \mathcal{R}_\text{emp}(c)$
3:     for $m \in \{1, \dots, M\}$ do
4:         $\tilde{r}^{[m](i)} = -\nabla_f L(y^{(i)}, f(\boldsymbol{x}^{(i)}))|_{f=\hat{f}^{[m-1]}}$, $\forall i \in \{1, \dots, n\}$
5:         for $l \in \{1, \dots, |\mathcal{B}|\}$ do
6:             $\hat{\boldsymbol{\theta}}_l^{[m]} = \left(\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{Z}_l + \boldsymbol{K}_l\right)^{-1} \boldsymbol{Z}_l^\mathsf{T}\tilde{\boldsymbol{r}}^{[m]}$
7:             $\text{SSE}_l = \sum_{i=1}^n (\tilde{r}^{[m](i)} - b_l(\boldsymbol{x}^{(i)}, \hat{\boldsymbol{\theta}}_l^{[m]}))^2$
8:         end for
9:         $l^{[m]} = \arg\min_{l\in\{1,\dots,|\mathcal{B}|\}} \text{SSE}_l$
10:        $\hat{f}^{[m]}(\boldsymbol{x}) = \hat{f}^{[m-1]}(\boldsymbol{x}) + \nu b_{l^{[m]}}(\boldsymbol{x}, \hat{\boldsymbol{\theta}}_{l^{[m]}}^{[m]})$
11:    end for
12:    return $\hat{f} = \hat{f}^{[M]}$
13: end procedure
```

2.3.3 Properties and Link to Generalized Additive Mixed Models

The estimated coefficients $\hat{\boldsymbol{\theta}}$ resulting from running the CWB algorithm are known to converge to the maximum likelihood solution (see, e.g., Schmid and Hothorn, 2008) for $M \to \infty$ under certain conditions. This is due to the fact that CWB performs a coordinate gradient descent update of a model defined by its additive base learners that exactly represent the structure of an additive mixed model (when defining the base learners according to Section 2.3.1) and by the objective function that corresponds to the negative (penalized) log-likelihood. Two important properties of this algorithm are 1) its coordinate-wise update routine, and 2) the nature of model updates using the L_2-loss. Due to the first property, CWB can be used in settings with $p \gg n$, as only a single additive term is fitted onto the pseudo-residuals in every iteration. This not only reduces the computational complexity of the algorithm for an increasing number of additive predictors (linear instead of quadratic) but also allows variable selection when stopping the routine early (e.g., based on a validation data set), as not all the additive components might have been selected into the model. In particular, this allows users to specify the full GAMM model without manual specification of the type of feature effect (fixed or random, linear or non-linear) and then automatically sparsify this model by an objective and data-driven feature selection. The second property, allows fitting models of the class of *generalized* linear/additive (mixed) models using only the L_2-loss instead of having to work with some iterative weighted least squares routine. In particular, this allows performing the proposed lossless distributed computations described in this paper, as we will discuss in Section 3.

2.4 Distributed Computing Setup and Privacy Protection

Before presenting our main results, we now introduce the distributed data setup we will work with throughout the remainder of this paper. The data set $\mathcal{D}$ is horizontally partitioned into S data sets $\mathcal{D}_s = \left\{\left(\boldsymbol{x}_s^{(1)}, y_s^{(1)}\right), \dots, \left(\boldsymbol{x}_s^{(n_s)}, y_s^{(n_s)}\right)\right\}$, $s = 1, \dots, S$ with n_s observations. Each data set $\mathcal{D}_s$ is located at a different site s and potentially follows a different data distributions $\mathbb{P}_{xy,s}$. The union of all data sets yields the whole data set $\mathcal{D} = \cup_{s=1}^{S} \mathcal{D}_s$ with mutually exclusive data sets $\mathcal{D}_s \cap \mathcal{D}_l = \emptyset \; \forall l, s \in \{1, \dots, S\}, l \neq s$. The vector of realizations per site is denoted by $\boldsymbol{y}_s \in \mathcal{Y}^{n_s}$.

In this distributed setup, multiple ways exist to communicate information without revealing individual information. More complex methods such as differential privacy (Dwork, 2006), homomorphic encryption (e.g., the Paillier cryptosystem; Paillier, 1999), or k-anonymity (Samarati and Sweeney, 1998) allow sharing information without violating an individual's privacy. An alternative option is to only communicate aggregated statistics. This is one of the most common approaches and is also used by `DataSHIELD` (Gaye et al, 2014) for GLMs or by Luo et al (2022) for LMMs. `DataSHIELD`, for example, uses a privacy level that indicates how many individual values must be aggregated to allow the communication of aggregated values. For example, setting the privacy level to a value of 5 enables sharing of summary statistics such as sums, means, variances, etc. if these are computed on at least 5 elements (observations).

Host and Site Setup

Throughout this article, we assume the $1, \dots, S$ sites or servers to have access to their respective data set $\mathcal{D}_s$. Each server is allowed to communicate with a host server that is also the analyst's machine. In this setting, the analyst can potentially see intermediate data used when running the algorithms, and hence each message communicated from the servers to the host must not allow any reconstruction of the original data. The host server is responsible for aggregating intermediate results and communicating these results back to the servers.

3 Distributed Component-Wise Boosting

We now present our distributed version of the CWB algorithm to fit privacy-preserving and lossless GAMMs. In the following, we first describe further specifications of our setup in Section 3.1, elaborate on the changes made to the set of base learners in Section 3.2, and then show how to adapt CWB's fitting routine in Section 3.3.

3.1 Setup

In the following, we distinguish between site-specific and shared effects. As effects estimated across sites typically correspond to fixed effects and effects modeled for each site separately are usually represented using random effects, we use the terms as synonyms in the following, i.e., *shared effects* and *fixed effects* are treated interchangeably and the same holds for *site-specific effects* and *random effects*. We note that this is only for ease of presentation and our approach also allows for site-specific fixed effects and random shared effects. As the data is not only located at different sites but also potentially follows different data distributions $\mathbb{P}_{xy,s}$ at each site s, we extend Eq. (1) to not only include random effects per site, but also site-specific smooth (random) effects $\phi_{j,s}(x_j)$, $s = 1,\dots,S$ for all features x_j with $j \in \mathcal{J}_3$. For every of these smooth effects $\phi_{j,s}$ we assume an existing shared effect $f_{j,\text{shared}}$ that is equal for all sites. These assumptions – particularly the choice of site-specific effects – are made for demonstration purposes. In a real-world application, the model structure can be defined individually to match the given data situation. However, note again that CWB intrinsically performs variable selection, and there is thus no need to manually define the model structure in practice. In order to incorporate the site information into the model, we add a variable $x_0^{(i)} \in \{1,\dots,S\}$ for the site to the data by setting $\tilde{\boldsymbol{x}}^{(i)} = (x_0^{(i)}, \boldsymbol{x}^{(i)})$. The site variable is a categorical feature with S classes.

3.2 Base Learners

For shared effects, we keep the original structure of CWB with base learners chosen from a set of possible learners $\mathcal{B}$. Section 3.3.1 explains how these shared effects are estimated in the distributed setup. We further define a regularized categorical base learner b_0 with basis transformation $g_0(x_0) = (\mathbb{1}_{\{1\}}(x_0),\dots,\mathbb{1}_{\{S\}}(x_0))^\mathsf{T}$ and design matrix $\boldsymbol{Z}_0 \in \mathbb{R}^{n\times S}$. We use b_0 to extend $\mathcal{B}$ with a second set of base learners $\mathcal{B}_\times = \{b_0 \times b \mid b \in \mathcal{B}\}$ to model site-specific random effects. All base learners in $\mathcal{B}_\times$ are row-wise tensor product base learners $b_{l_\times} = b_0 \times b_l$ of the regularized categorical base learner b_0 dummy-encoding every site and all other existing base learners $b_l \in \mathcal{B}$. This allows for potential inclusion of random effects for every fixed effect in the model. More specifically, the l^{th} site-specific effect given by the row-wise tensor product base learner $b_{l_\times}$ uses the basis transformation $g_{l_\times} = g_0 \otimes g_l$

$$g_{l_\times}(\tilde{\boldsymbol{x}}) = g_0(x_0)^\mathsf{T} \otimes g_l(\boldsymbol{x})^\mathsf{T} \\ = (\underbrace{\mathbb{1}_{\{1\}}(x_0) g_l(\boldsymbol{x})^\mathsf{T}}_{=g_{l_\times,1}},\dots,\underbrace{\mathbb{1}_{\{S\}}(x_0) g_l(\boldsymbol{x})^\mathsf{T}}_{=g_{l_\times,S}})^\mathsf{T}, \tag{2}$$

where the basis transformation g_l is equal for all S sites. After distributed computation (see Eq. (4) in the next section), the estimated coefficients are $\hat{\boldsymbol{\theta}}_{l_\times} = (\hat{\boldsymbol{\theta}}_{l_\times,1}^\mathsf{T},\dots,\hat{\boldsymbol{\theta}}_{l_\times,S}^\mathsf{T})^\mathsf{T}$ with $\hat{\boldsymbol{\theta}}_{l_\times,s} \in \mathbb{R}^{d_l}$. The regularization of the row-wise Kronecker base learners not only controls their flexibility but also assures identifiable when additionally including a shared (fixed) effect for the same covariate. The penalty matrix $\boldsymbol{K}_{l_\times} = \lambda_0 \boldsymbol{K}_0 \otimes \boldsymbol{I}_{d_l} + \boldsymbol{I}_S \otimes \lambda_{l_\times} \boldsymbol{K}_l \in \mathbb{R}^{Sd_l\times Sd_l}$ is given as Kronecker sum of the penalty matrix of the categorical site effect and the penalty matrices $\boldsymbol{K}_0$ and $\boldsymbol{K}_l$ with respective regularization strengths $\lambda_0, \lambda_{l_\times}$. As $\boldsymbol{K}_0 = \lambda_0 \boldsymbol{I}_S$ is a diagonal matrix, $\boldsymbol{K}_{l_\times}$ is a block matrix with entries $\lambda_0 \boldsymbol{I}_{d_l} + \lambda_{l_\times} \boldsymbol{K}_l$ on the diagonal blocks. Moreover, as g_0 is a binary vector, we can also express the design matrix $\boldsymbol{Z}_{l_\times} \in \mathbb{R}^{n\times Sd_l}$ as a block matrix, yielding

$$\boldsymbol{Z}_{l_\times} = \mathsf{diag}(\boldsymbol{Z}_{l,1},\dots,\boldsymbol{Z}_{l,S}),\ \boldsymbol{K}_{l_\times} \\ = \mathsf{diag}(\lambda_0 \boldsymbol{I}_{d_l} + \lambda_{l_\times}\boldsymbol{K}_l,\dots,\lambda_0 \boldsymbol{I}_{d_l} + \lambda_{l_\times}\boldsymbol{K}_l), \tag{3}$$

where $\boldsymbol{Z}_{l,k}$ are the distributed design matrices of b_l on sites $s = 1,\dots,S$.

3.3 Fitting Algorithm

We now describe the adaptions required to allow for distributed computations of the CWB fitting routine. In Sections 3.3.1 and 3.3.2, we show the equality between our distributed fitting approach and CWB fitted on pooled data. Section 3.3.3 describes the remaining details such as distributed SSE calculations, distributed model updates, and pseudo residual updates in the distributed setup.

Section 3.4 summarizes the distributed CWB algorithm and Section 3.5 elaborates on the communication costs of our algorithm.

3.3.1 Distributed Shared Effects Computation

Fitting CWB in a distributed fashion requires adapting the fitting process of the base learner b_l in Algorithm 1 to distributed data. To allow for shared effects computations across different sites without jeopardizing privacy, we take advantage of CWB's update scheme, which boils down to a (penalized) least squares estimation per iteration for every base learner. This allows us to build upon existing work such as Karr et al (2005) to fit linear models in a distributed fashion by just communicating aggregated statistics between sites and the host.

In a first step, the aggregated matrices $\boldsymbol{F}_{l,s} = \boldsymbol{Z}_{l,s}^\mathsf{T}\boldsymbol{Z}_{l,s}$ and vectors $\boldsymbol{u}_{l,s} = \boldsymbol{Z}_{l,s}^\mathsf{T}\boldsymbol{y}_s$ are computed on each site. In our privacy setup (Section 2.4), communicating $\boldsymbol{F}_{l,s}$ and $\boldsymbol{u}_{l,s}$ is allowed as long as the privacy-aggregation level per site is met. In a second step, the site information is aggregated to a global information $\boldsymbol{F}_l = \sum_{s=1}^S \boldsymbol{F}_{l,s} + \boldsymbol{K}_l$ and $\boldsymbol{u}_l = \sum_{s=1}^S \boldsymbol{u}_{l,s}$ and then used to estimate the model parameters $\hat{\boldsymbol{\theta}}_l = \boldsymbol{F}_l^{-1}\boldsymbol{u}_l$. This approach, referred to as distFit, is explained again in detail in Algorithm 2 and used for the shared effect computations of the model by substituting $\hat{\boldsymbol{\theta}}_l^{[m]} = \left(\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{Z}_l + \boldsymbol{K}_l\right)^{-1}\boldsymbol{Z}_l^\mathsf{T}\tilde{\boldsymbol{r}}^{[m]}$ (Algorithm 1 line 6) with $\hat{\boldsymbol{\theta}}_l^{[m]} = \mathsf{distFit}(\boldsymbol{Z}_{l,1}, \dots, \boldsymbol{Z}_{l,S}, \tilde{\boldsymbol{r}}_1^{[m]}, \dots, \tilde{\boldsymbol{r}}_S^{[m]}, \boldsymbol{K}_l)$.

Note that the pseudo residuals $\tilde{\boldsymbol{r}}_k^{[m]}$ are also securely located at each site and are updated after each iteration. Details about the distributed pseudo residuals updates are explained in Section 3.3.3. We also note that the computational complexity of fitting CWB can be drastically reduced by pre-calculating and storing $(\boldsymbol{Z}_l^\mathsf{T}\boldsymbol{Z}_l + \boldsymbol{K}_l)^{-1}$ in a first initialization step, as the matrix is independent of iteration m, and reusing these pre-calculated matrices in all subsequent iterations (cf. Schalk et al, 2022a). Using pre-calculated matrices also reduces the amount of required communication between sites and host.

Algorithm 2 Distributed Effect Estimation. The line prefixes [S] and [H] indicate whether the operation is conducted at the sites ([S]) or at the host ([H]).

```
Input  Sites design matrices $\boldsymbol{Z}_{l,1}, \dots, \boldsymbol{Z}_{l,S}$,
       response vectors $\boldsymbol{y}_1, \dots, \boldsymbol{y}_S$ and
       an optional penalty matrix $\boldsymbol{K}_l$.
Output Estimated parameter vector $\hat{\boldsymbol{\theta}}_l$.
1:  procedure distFit($\boldsymbol{Z}_{l,1}, \dots, \boldsymbol{Z}_{l,S}, \boldsymbol{y}_1, \dots, \boldsymbol{y}_S, \boldsymbol{K}_l$)
2:      for $s \in \{1, \dots, S\}$ do
3:          [S] $\boldsymbol{F}_{l,s} = \boldsymbol{Z}_{l,s}^\mathsf{T}\boldsymbol{Z}_{l,s}$
4:          [S] $\boldsymbol{u}_{l,s} = \boldsymbol{Z}_{l,s}^\mathsf{T}\boldsymbol{y}_s$
5:          [S] Communicate $\boldsymbol{F}_{l,s}$ and $\boldsymbol{u}_{l,s}$ to the host
6:      end for
7:      [H] $\boldsymbol{F}_l = \sum_{s=1}^S \boldsymbol{F}_{l,s} + \boldsymbol{K}_l$
8:      [H] $\boldsymbol{u}_l = \sum_{s=1}^S \boldsymbol{u}_{l,s}$
9:      [H] return $\hat{\boldsymbol{\theta}}_l = \boldsymbol{F}_l^{-1}\boldsymbol{u}_l$
10: end procedure
```

3.3.2 Distributed Site-specific Effects Computation

If we pretend that the fitting of the base learner $b_{l_\times}$ is performed on the pooled data, we obtain

$$\hat{\boldsymbol{\theta}}_{l_\times} = \left(\boldsymbol{Z}_{l_\times}^\mathsf{T}\boldsymbol{Z}_{l_\times} + \boldsymbol{K}_{l_\times}\right)^{-1}\boldsymbol{Z}_{l_\times}^\mathsf{T}\boldsymbol{y} = \begin{pmatrix} (\boldsymbol{Z}_{l,1}^\mathsf{T}\boldsymbol{Z}_{l,1} + \lambda_0\boldsymbol{I}_{d_l} + \boldsymbol{K}_l)^{-1}\boldsymbol{Z}_{l,1}^\mathsf{T}\boldsymbol{y}_1 \\ \vdots \\ (\boldsymbol{Z}_{l,S}^\mathsf{T}\boldsymbol{Z}_{l,S} + \lambda_0\boldsymbol{I}_{d_l} + \boldsymbol{K}_l)^{-1}\boldsymbol{Z}_{l,S}^\mathsf{T}\boldsymbol{y}_S \end{pmatrix}, \tag{4}$$

where (4) is due to the block structure, as described in (3) of Section 3.2. This shows that the fitting of the site-specific effects $\hat{\boldsymbol{\theta}}_{l_\times}$ can be split up into the fitting of individual parameters

$$\hat{\boldsymbol{\theta}}_{l_\times,s} = (\boldsymbol{Z}_{l,s}^\mathsf{T}\boldsymbol{Z}_{l,s} + \lambda_0\boldsymbol{I}_{d_l} + \boldsymbol{K}_l)^{-1}\boldsymbol{Z}_{l,s}^\mathsf{T}\boldsymbol{y}_s. \tag{5}$$

It is thus possible to compute site-specific effects at the respective site without the need to share any information with the host. The host, in turn, only requires the SSE of the respective base learner (see next Section 3.3.3) to perform the next iteration of CWB. Hence, during the fitting process, the parameter estimates remain at their sites and are just updated if the site-specific base learner is selected. This again minimizes the amount of data communication between sites and host and speeds up the fitting process. After the fitting phase, the aggregated site-specific parameters are communicated once in a last communication step to obtain the final model.

3.3.3 Pseudo Residual Updates, SSE Calculation, and Base Learner Selection

The remaining challenges to run the distributed CWB algorithm are 1) the pseudo residual calculation (Algorithm 1 line 4), 2) the SSE calculation (Algorithm 1 line 7), and 3) base learner selection (Algorithm 1 line 9).

Distributed pseudo residual updates

The site-specific response vector $\boldsymbol{y}_s$ containing the values $y^{(i)}$, $i \in \{1,\dots,n_s\}$ is the basis of the pseudo residual calculation. We assume that every site s has access to all shared effects as well as the site-specific information of all site-specific base learners $b_{l_\times}$ only containing the respective parameters $\hat{\boldsymbol{\theta}}_{l_\times,s}$. Based on these base learners, it is thus possible to compute a *site model* $\hat{f}_s^{[m]}$ as a representative of $\hat{f}^{[m]}$ on every site s. The pseudo residual updates $\tilde{\boldsymbol{r}}_s^{[m]}$ per site are then based on $\hat{f}_s^{[m]}$ via $\tilde{r}_s^{[m](i)} = -\nabla_f L(y^{(i)}, f(\boldsymbol{x}^{(i)}))|_{f=\hat{f}_s^{[m-1]}}$, $i \in \{1,\dots,n_s\}$ using $\mathcal{D}_s$. Most importantly, all remaining steps of the distributed CWB fitting procedure do not share the pseudo residuals $\tilde{\boldsymbol{r}}_s^{[m]}$ in order to avoid information leakage about $\boldsymbol{y}_s$.

Distributed SSE calculation and base learner selection

After fitting all base learners $b_l \in \mathcal{B}$ and $b_{l_\times} \in \mathcal{B}_\times$ to $\tilde{\boldsymbol{r}}_s^{[m]}$, we obtain $\hat{\boldsymbol{\theta}}_l^{[m]}$, $l = 1,\dots,|\mathcal{B}|$, and $\hat{\boldsymbol{\theta}}_{l_\times}^{[m]}$, $l_\times = 1_\times,\dots,|\mathcal{B}_\times|$. Calculating the SSE distributively for the l^{th} and ${l_\times}^{\text{th}}$ base learner b_l and $b_{l_\times}$, respectively, requires calculating $2S$ site-specific SSE values:

$$\begin{aligned}\mathsf{SSE}_{l,s} &= \sum_{i=1}^{n_s}\left(\tilde{r}_s^{[m](i)} - b_l(\boldsymbol{x}_s^{(i)}, \hat{\boldsymbol{\theta}}_l^{[m]})\right)^2\\ &= \sum_{i=1}^{n_s}(\tilde{r}^{[m](i)} - g_l(\boldsymbol{x}^{(i)})^\mathsf{T}\hat{\boldsymbol{\theta}}_l^{[m]})^2,\\ \mathsf{SSE}_{l_\times,s} &= \sum_{i=1}^{n_s}\left(\tilde{r}_s^{[m](i)} - b_{l_\times}(\boldsymbol{x}_s^{(i)}, \hat{\boldsymbol{\theta}}_{l_\times}^{[m]})\right)^2\\ &= \sum_{i=1}^{n_s}(\tilde{r}_s^{[m](i)} - g_l(\boldsymbol{x}^{(i)})^\mathsf{T}\hat{\boldsymbol{\theta}}_{l_\times,s}^{[m]})^2.\end{aligned}$$

The site-specific SSE values are then sent to the host and aggregated to $\mathsf{SSE}_l = \sum_{s=1}^S \mathsf{SSE}_{l,s}$. If privacy constraints have been met in all previous calculations, sharing the individual SSE values is not critical and does not violate any privacy constraints as the value is an aggregation of all n_s observations for all sites s.

Having gathered all SSE values at the host location, selecting the best base learner in the current iteration is done in the exact same manner as for the non-distributed CWB algorithm by selecting $l^{[m]} = \operatorname{arg\,min}_{l\in\{1,\dots,|\mathcal{B}|,1_\times,\dots,|\mathcal{B}|_\times\}} \mathsf{SSE}_l$. After the selection, the index $l^{[m]}$ is shared with all sites to enable the update of the site-specific models $\hat{f}_s^{[m]}$. If a shared effect is selected, the parameter vector $\hat{\boldsymbol{\theta}}_{l^{[m]}}^{[m]}$ is shared with all sites. Caution must be taken when the number of parameters of one base learner is equal to the number of observations, as this allows reverse-engineering private data. In the case of a site-specific effect selection, no parameter needs to be communicated, as the respective estimates are already located at each site.

3.4 Distributed CWB Algorithm with Site-Specific Effects

Assembling all pieces, our distributed CWB algorithm is summarized in Algorithm 3.

3.5 Communication Costs

While the CWB iterations themselves can be performed in parallel on every site and do not slow down the process compared to a pooled calculation, it is worth discussing the communication costs of distrCWB. During the initialization, data is shared just once, while the fitting phase requires the communication of data in each iteration. Let $d = \max_l d_l$ be the maximum number of basis functions (or, alternatively, assume d basis functions for all base learners). The two main drivers of the communication costs are the number of boosting iterations M and the number of base learners $|\mathcal{B}|$. Because of the iterative nature of CWB with a single loop over the boosting iterations, the communication costs (both for the host and each site) scale linearly with the number of boosting

Algorithm 3 Distributed CWB Algorithm.
The line prefixes `[S]` and `[H]` indicate whether the operation is conducted at the sites (`[S]`) or at the host (`[H]`).

Input Sites with site data $\mathcal{D}_k$, learning rate ν, number of boosting iterations M, loss function L, set of shared effects $\mathcal{B}$ and respective site-specific effects $\mathcal{B}_\times$
Output Prediction model $\hat{f}$

```
procedure distrCWB($\nu, L, \mathcal{B}, \mathcal{B}_\times$)
    Initialization:
    [H] Initialize shared model $\hat{f}^{[0]}_{\text{shared}}(\boldsymbol{x}) = \operatorname{arg\,min}_{c\in\mathbb{R}} \mathcal{R}_{\text{emp}}(c)$
    [S] Calculate $\boldsymbol{Z}_{l,s}$ and $\boldsymbol{F}_{l,s} = \boldsymbol{Z}^{\mathsf{T}}_{l,s}\boldsymbol{Z}_{l,s}$, $\forall l \in \{1,\dots,|\mathcal{B}|\},\ s \in \{1,\dots,S\}$
    [S] Set $\hat{f}^{[0]}_s = \hat{f}^{[0]}_{\text{shared}}$
    for $m \in \{1,\dots,M\}$ or while an early stopping criterion is not met do
        [S] Update pseudo residuals:
        [S]    $\tilde{r}^{[m](i)}_s = -\nabla_f L(y^{(i)}, f(\boldsymbol{x}^{(i)}))|_{f=\hat{f}^{[m-1]}_s}$,  $\forall i \in \{1,\dots,n_s\}$
        for $l \in \{1,\dots,|\mathcal{B}|\}$ do
            [H] Calculate shared effect: $\hat{\boldsymbol{\theta}}^{[m]}_l = \mathsf{distFit}(\boldsymbol{Z}_{l,1},\dots,\boldsymbol{Z}_{l,S},\boldsymbol{y}_1,\dots,\boldsymbol{y}_S,\boldsymbol{K}_l)$
            [H] Communicate $\hat{\boldsymbol{\theta}}^{[m]}_l$ to the sites
            for $k \in \{1,\dots,S\}$ do
                [S] Fit $l^{\text{th}}$ site-specific effect: $\hat{\boldsymbol{\theta}}^{[m]}_{l_\times,s} = (\boldsymbol{F}_{l,s} + \lambda_0 \boldsymbol{I}_{d_l} + \boldsymbol{K}_l)^{-1}\boldsymbol{Z}_{l,s}\tilde{\boldsymbol{r}}^{[m]}_s$
                [S] Calculate the SSE for the $l^{\text{th}}$ shared and site-specific effect:
                [S]    $\mathsf{SSE}_{l,s} = \sum_{i=1}^{n_s}(\tilde{r}^{[m](i)} - g_l(\boldsymbol{x}^{(i)})^{\mathsf{T}}\hat{\boldsymbol{\theta}}^{[m]}_l)^2$
                [S]    $\mathsf{SSE}_{l_\times,s} = \sum_{i=1}^{n_s}(\tilde{r}^{[m](i)}_s - g_l(\boldsymbol{x}^{(i)})^{\mathsf{T}}\hat{\boldsymbol{\theta}}^{[m]}_{l_\times,s})^2$
                [S] Send $\mathsf{SSE}_{l,s}$ and $\mathsf{SSE}_{l_\times,s}$ to the host
            end for
            [H] Aggregate SSE values: $\mathsf{SSE}_l = \sum_{s=1}^{S}\mathsf{SSE}_{l,s}$ and $\mathsf{SSE}_{l_\times} = \sum_{s=1}^{S}\mathsf{SSE}_{l_\times,s}$
        end for
        [H] Select best base learner: $l^{[m]} = \operatorname{arg\,min}_{l\in\{1,\dots,|\mathcal{B}|,1_\times,\dots,|\mathcal{B}|_\times\}}\mathsf{SSE}_l$
        if $b_{l^{[m]}}$ is a shared effect then
            [H] Update model: $\hat{f}^{[m]}_{\text{shared}}(\boldsymbol{x}) = \hat{f}^{[m-1]}_{\text{shared}}(\boldsymbol{x}) + \nu b_{l^{[m]}}(\boldsymbol{x}, \hat{\boldsymbol{\theta}}^{[m]}_{l^{[m]}})$
            [H] Upload model update $\hat{\boldsymbol{\theta}}^{[m]}_{l^{[m]}}$ to the sites.
        end if
        [S] Update site model $\hat{f}^{[m]}_s$ via parameter updates $\hat{\boldsymbol{\theta}}_{l^{[m]}} = \hat{\boldsymbol{\theta}}_{l^{[m]}} + \nu\hat{\boldsymbol{\theta}}^{[m]}_{l^{[m]}}$
    end for
    [S] Communicate site-specific effects $\hat{\boldsymbol{\theta}}_{1_\times},\dots,\hat{\boldsymbol{\theta}}_{|\mathcal{B}|_\times}$ to the host
    [H] Add site-specific effects to the model of shared effects $\hat{f}^{[M]}_{\text{shared}}$ to obtain the full model $\hat{f}^{[M]}$
    [H] return $\hat{f} = \hat{f}^{[M]}$
end procedure
```

iterations M, i.e., $\mathcal{O}(M)$. For the analysis of communication costs in terms of the number of base learners, we distinguish between the initialization phase and the fitting phase.

Initialization

As only the sites share $\boldsymbol{F}_{l,s} \in \mathbb{R}^{d\times d}$, $\forall l \in \{1,\dots,|\mathcal{B}|\}$, the transmitted amount of values is $d^2|\mathcal{B}|$ for each site and therefore scales linearly with $|\mathcal{B}|$, i.e., $\mathcal{O}(|\mathcal{B}|)$. The host does not communicate any values during the initialization.

Fitting

In each iteration, every site shares its vector $\boldsymbol{Z}^{\mathsf{T}}_{l,s}\tilde{\boldsymbol{r}}^{[m]}_s \in \mathbb{R}^d$, $\forall l \in \{1,\dots,|\mathcal{B}|\}$. Over the course of M boosting iterations, each site therefore shares $dM|\mathcal{B}|$ values. Every site also communicates the SSE values, i.e., 2 values (index and SSE value) for every base learner and thus $2M|\mathcal{B}|$ values for all iterations and base learners. In total, each site communicates $M|\mathcal{B}|(d+2)$ values. The communication costs for all sites are therefore $\mathcal{O}(|\mathcal{B}|)$. The host, in turn, communicates the estimated parameters $\hat{\boldsymbol{\theta}}^{[m]} \in \mathbb{R}^d$ of the $|\mathcal{B}|$ shared effects. Hence, $dM|\mathcal{B}|$ values as well as the index of the best base learner in each iteration are transmitted. In total, the host therefore communicates $dM|\mathcal{B}|+M$ values to the sites, and costs are therefore also $\mathcal{O}(|\mathcal{B}|)$.

4 Application

We now showcase our algorithm on a heart disease data set that consists of patient data gathered all over the world. The data were collected at four different sites by the 1) Hungarian Institute of Cardiology, Budapest (Andras Janosi, M.D.), 2) University Hospital, Zurich, Switzerland (William Steinbrunn, M.D.), 3) University Hospital, Basel, Switzerland (Matthias Pfisterer, M.D.), and 4) V.A. Medical Center, Long Beach, and Cleveland Clinic Foundation (Robert Detrano, M.D., Ph.D.), and is thus suited for a multi-site distributed analysis. The individual data sets are freely available at https://archive.ics.uci.edu/ml/datasets/heart+disease (Dua and Graff, 2017). For our analysis, we set the privacy level (cf. Section 2.4) to 5 which is a common default.

4.1 Data Description

The raw data set contains 14 covariates, such as the chest pain type (`cp`), resting blood pressure (`trestbps`), maximum heart rate (`thalach`), sex, exercise-induced angina (`exang`), or ST depression (i.e., abnormal difference of the ST segment from the baseline on an electrocardiogram) induced by exercise relative to rest (`oldpeak`). A full list of covariates and their abbreviations is given on the data set's website. After removing non-informative covariates and columns with too many missing values at each site, we obtain $n_{\text{cleveland}} = 303$, $n_{\text{hungarian}} = 292$, $n_{\text{switzerland}} = 116$, and $n_{\text{va}} = 140$ observations and 8 covariates. A table containing the description of the abbreviations of these covariates is given in Table 1 in the Supplementary Material B.1. For our application, we assume that missing values are completely at random and all data sets are exclusively located at each sites. The task is to determine important risk factors for heart diseases. The target variable is therefore a binary outcome indicating the presence of heart disease or not.

4.2 Analysis and Results

We follow the practices to setup CWB as mentioned in Section 2.3.2 and run the distributed CWB algorithm with a learning rate of 0.1 and a maximum number of 100000 iterations. To determine an optimal stopping iteration for CWB, we use 20 % of the data as validation data and set the patience to 5 iterations. In other words, the algorithm stops if no risk improvement on the validation data is observed in 5 consecutive iterations. For the numerical covariates, we use a P-spline with 10 cubic basis functions and second-order difference penalties. All base learners are penalized accordingly to a global degree of freedom that we set to 2.2 (to obtain unbiased feature selection) while the random intercept is penalized according to 3 degrees of freedom (see the Supplementary Material B.2 for more details). Since we are modelling a binary response variable, h^{-1} is the inverse logit function $\mathsf{logit}^{-1}(f) = (1+\mathsf{exp}(-f))^{-1}$. The model for an observation of site s, conditional on its random effects $\boldsymbol{\gamma}$, is given in the Supplementary Material B.3.

Results

The algorithm stops after $m_{\text{stop}} = 5578$ iterations as the risk on the validation data set starts to increase (cf. Figure 1 in the Supplementary Material B.4). Out of these 5578 iterations, the distributed CWB algorithm selects a shared effect in 782 iterations and site-specific effects in 4796 iterations. This indicates that the data is rather heterogeneous and requires site-specific (random) effects. Figure 1 (Left) shows traces of how and when the different additive terms (base learners) entered the model during the fitting process and illustrates the selection process of CWB.

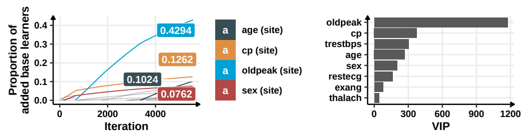

Fig. 1: Left: Model trace showing how and when the four most selected additive terms entered the model. Right: Variable importance (cf. Au et al, 2019) of selected features in decreasing order.

The estimated effect of the most important feature `oldpeak` (cf. Figure 1, Right) found is further visualized in Figure 2. Looking at the shared effect, we find a negative influence on the risk of heart disease when increasing ST depression (`oldpeak`). When accounting for site-specific deviations, the effect becomes more diverse, particularly for Hungary.

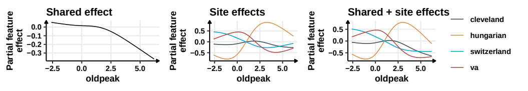

Fig. 2: Decomposition of the effect of `oldpeak` into the shared (left) and the site-specific effects (middle). The plot on the right-hand side shows the sum of shared and site-specific effects.

In the Supplementary Material B.5 and B.6, we provide the partial effects for all features and showcase the conditional predictions of the fitted GAMM model for a given site.

Comparison of Estimation Approaches

The previous example shows partial feature effects that exhibit shrinkage due to the early stopping of CWB's fitting routine. While this prevents overfitting and induces a sparse model, we can also run CWB for a very large amount of iterations without early stopping to approximate the unregularized and hence unbiased maximum likelihood solution. We illustrate this in the following by training CWB and our distributed version for 100000 iterations and compare its partial effects to the ones of a classical mixed model-based estimation routine implemented in the `R` package `mgcv` (Wood, 2017).

Results of the estimated partial effects of our distributed CWB algorithm and the original CWB on pooled data show a perfect overlap (cf. Figure 3). This again underpins the lossless property of the proposed algorithm. The site-specific effects on the pooled data are fitted by defining a row-wise Kronecker base learner for all features and the site as a categorical variable. The same approach is used to estimate a GAMM using `mgcv` fitted on the pooled data with tensor products between the main feature and the categorical site variable. A comparison of all partial feature effects is given in the Supplementary Material B.7 showing good alignment between the different methods. For the `oldpeak` effect shown in Figure 3, we also see that the partial effects of the two CWB methods are very close to the mixed model-based estimation, with only smaller differences caused by a slightly different penalization strength of both approaches. The empirical risk is 0.4245 for our distributed CWB algorithm, 0.4245 for CWB on the pooled data, and 0.4441 for the GAMM on the pooled data.

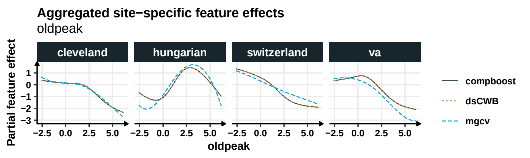

Fig. 3: Comparison of the site-specific effects for `oldpeak` between the distributed (`dsCWB`) and pooled CWB approach (`compboost`) as well as estimates of from `mgcv`.

5 Discussion

We proposed a novel algorithm for distributed, lossless, and privacy-preserving GAMM estimation to analyze horizontally partitioned data. To account for data heterogeneity of different sites we introduced site-specific (smooth) random effects. Using CWB as the fitting engine allows estimation in high-dimensional settings and fosters variable as well as effect selection. This also includes a data-driven selection of shared and site-specific features, providing additional data insights. Owing to the flexibility of boosting and its base learners, our algorithm is easy to extend and can also account for interactions, functional regression settings (Brockhaus et al, 2020), or modeling survival tasks (Bender et al, 2020).

An open challenge for the practical use of our approach is its high communication costs. For larger iterations (in the 10 or 100 thousands), computing a distributed model can take several hours. One option to reduce the total runtime is to incorporate accelerated optimization recently proposed in Schalk et al (2022a). Another driver that influences the runtime is the latency of the technical setup. Future improvements could reduce the number of communications, e.g., via multiple fitting rounds at the different sites before communicating the intermediate results.

A possible future extension of our approach is to account for both horizontally and vertically distributed data. Since the algorithm is performing component-wise (coordinate-wise) updates, the extension to vertically distributed data naturally falls into the scope of its fitting procedure. This would, however, require a further advanced technical setup and the need to ensure consistency across sites.

Declarations

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Anjum MM, Mohammed N, Li W, et al (2022) Privacy preserving collaborative learning of generalized linear mixed model. Journal of Biomedical Informatics 127:104,008

Au Q, Schalk D, Casalicchio G, et al (2019) Component-wise boosting of targets for multi-output prediction. arXiv preprint arXiv:190403943

Augustyn DR, Wyciślik Ł, Mrozek D (2021) Perspectives of using cloud computing in integrative analysis of multi-omics data. Briefings in Functional Genomics 20(4):198–206. https://doi.org/10.1093/bfgp/elab007, URL https://doi.org/10.1093/bfgp/elab007

Bazeley P (2012) Integrative analysis strategies for mixed data sources. American Behavioral Scientist 56(6):814–828

Bender A, Rügamer D, Scheipl F, et al (2020) A general machine learning framework for survival analysis. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, pp 158–173

Boyd K, Lantz E, Page D (2015) Differential privacy for classifier evaluation. In: Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, pp 15–23

Brockhaus S, Rügamer D, Greven S (2020) Boosting functional regression models with fdboost. Journal of Statistical Software 94(10):1 – 50

Brumback BA, Ruppert D, Wand MP (1999) Variable selection and function estimation in additive nonparametric regression using a data-based prior: Comment. Journal of the American Statistical Association 94(447):794–797

Bühlmann P, Yu B (2003) Boosting with the L2 loss: regression and classification. Journal of the American Statistical Association 98(462):324–339

Bühlmann P, Hothorn T, et al (2007) Boosting algorithms: Regularization, prediction and model fitting. Statistical science 22(4):477–505

Chen YR, Rezapour A, Tzeng WG (2018) Privacy-preserving ridge regression on distributed data. Information Sciences 451:34–49

Curran PJ, Hussong AM (2009) Integrative data analysis: the simultaneous analysis of multiple data sets. Psychological methods 14(2):81

Dua D, Graff C (2017) UCI machine learning repository. URL http://archive.ics.uci.edu/ml

Dwork C (2006) Differential privacy. In: International Colloquium on Automata, Languages, and Programming, Springer, pp 1–12

Eilers PH, Marx BD (1996) Flexible smoothing with B-splines and penalties. Statistical science pp 89–102

Gambs S, Kégl B, Aïmeur E (2007) Privacy-preserving boosting. Data Mining and Knowledge Discovery 14(1):131–170

Gaye A, Marcon Y, Isaeva J, et al (2014) Datashield: taking the analysis to the data, not the data to the analysis. International journal of epidemiology 43(6):1929–1944

Hoerl AE, Kennard RW (1970) Ridge regression: Biased estimation for nonorthogonal problems. Technometrics 12(1):55–67

Hofner B, Hothorn T, Kneib T, et al (2011) A framework for unbiased model selection based on boosting. Journal of Computational and Graphical Statistics 20(4):956–971

Jones EM, Sheehan NA, Gaye A, et al (2013) Combined analysis of correlated data when data cannot be pooled. Stat 2(1):72–85

Karr AF, Lin X, Sanil AP, et al (2005) Secure regression on distributed databases. Journal of Computational and Graphical Statistics 14(2):263–279

Lazarevic A, Obradovic Z (2001) The distributed boosting algorithm. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pp 311–316

Li J, Kuang X, Lin S, et al (2020a) Privacy preservation for machine learning training and classification based on homomorphic encryption schemes. Information Sciences 526:166–179

Li Q, Wen Z, He B (2020b) Practical federated gradient boosting decision trees. In: Proceedings of the AAAI conference on artificial intelligence, pp 4642–4649

Liew BX, Rügamer D, Abichandani D, et al (2020) Classifying individuals with and without patellofemoral pain syndrome using ground force profiles – Development of a method using functional data boosting. Gait & Posture 80:90–95

Litwin W, Mark L, Roussopoulos N (1990) Interoperability of multiple autonomous databases. ACM Computing Surveys (CSUR) 22(3):267–293

Lu CL, Wang S, Ji Z, et al (2015) Webdisco: a web service for distributed cox model learning without patient-level data sharing. Journal of the American Medical Informatics Association 22(6):1212–1219

Luo C, Islam M, Sheils NE, et al (2022) Dlmm as a lossless one-shot algorithm for collaborative multi-site distributed linear mixed models. Nature Communications 13(1):1–10

McCullagh P, Nelder JA (2019) Generalized linear models. Routledge

McMahan B, Moore E, Ramage D, et al (2017) Communication-efficient learning of deep networks from decentralized data. In: Artificial intelligence and statistics, PMLR, pp 1273–1282

Mirza B, Wang W, Wang J, et al (2019) Machine learning and integrative analysis of biomedical big data. Genes 10(2):87

Mohassel P, Zhang Y (2017) Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE symposium on security and privacy (SP), IEEE, pp 19–38

Naehrig M, Lauter K, Vaikuntanathan V (2011) Can homomorphic encryption be practical? In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop, pp 113–124

Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In: International conference on the theory and applications of cryptographic techniques, Springer, pp 223–238

R Core Team (2021) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL https://www.R-project.org/

Rügamer D, Brockhaus S, Gentsch K, et al (2018) Boosting factor-specific functional historical models for the detection of synchronization in bioelectrical signals. Journal of the Royal Statistical Society: Series C

(Applied Statistics) 67(3):621–642. https://doi.org/https://doi.org/10.1111/rssc.12241, URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssc.12241

Saintigny P, Zhang L, Fan YH, et al (2011) Gene expression profiling predicts the development of oral cancer. Cancer Prevention Research 4(2):218–229

Samarati P, Sweeney L (1998) Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression

Schalk D, Bischl B, Rügamer D (2022a) Accelerated componentwise gradient boosting using efficient data representation and momentum-based optimization. Journal of Computational and Graphical Statistics 0(ja):1–27. https://doi.org/10.1080/10618600.2022.2116446, URL https://doi.org/10.1080/10618600.2022.2116446

Schalk D, Hoffmann VS, Bischl B, et al (2022b) Distributed non-disclosive validation of predictive models by a modified roc-glm. arXiv preprint arXiv:220310828

Schmid M, Hothorn T (2008) Boosting additive models using component-wise p-splines. Computational Statistics & Data Analysis 53(2):298–311

Ünal AB, Pfeifer N, Akgün M (2021) ppaurora: Privacy preserving area under receiver operating characteristic and precision-recall curves with secure 3-party computation. ArXiv 2102

Wood SN (2017) Generalized additive models: an introduction with R. Chapman and Hall/CRC

Wu Y, Jiang X, Kim J, et al (2012) G rid binary lo gistic re gression (glore): building shared models without sharing data. Journal of the American Medical Informatics Association 19(5):758–764

Yan Z, Zachrison KS, Schwamm LH, et al (2022) Fed-glmm: A privacy-preserving and computation-efficient federated algorithm for generalized linear mixed models to analyze correlated electronic health records data. medRxiv

Zhu R, Jiang C, Wang X, et al (2020) Privacy-preserving construction of generalized linear mixed model for biomedical computation. Bioinformatics 36(Supplement_1):i128–i135. https://doi.org/10.1093/bioinformatics/btaa478, URL https://doi.org/10.1093/bioinformatics/btaa478

CHAPTER 10

Distributed non-disclosive validation of predictive models by a modified ROC-GLM

Contributing article

Schalk, D., Hoffmann, V. S., Bischl, B., and Mansmann, U. (2022b). Distributed non-disclosive validation of predictive models by a modified roc-glm. *arXiv preprint arXiv:2203.10828*

At the time the thesis was handed in, the article was in review at the journal *BMC Medical Research Methodology*.

Declaration of contributions

Daniel Schalk wrote the manuscript, implemented the methods and the simulation study, and prepared the use case. He also created all graphics and the interpretation of the results from the simulation study and from the use case. In addition, much effort was put into reproducibility, for which he created a Docker container and a GitHub repository with the simulation study results. The application example is automatically executed via GitHub every week to ensure functionality.

Contribution of the coauthors

The idea of the distributed AUC calculation originated from Prof. Dr. Ulrich Mansmann. All coauthors provided substantial assistance in writing the manuscript and interpreting the simulation study results.

Distributed non-disclosive validation of predictive models by a modified ROC-GLM

Daniel Schalk[1,3,4*], Verena S. Hoffmann[2,3], Bernd Bischl[1,4] and Ulrich Mansmann[1,2,3]

*Correspondence: daniel.schalk@stat.uni-muenchen.de

[1]Department of Statistics, LMU Munich, Munich, Germany

Full list of author information is available at the end of the article

Abstract

Background: Distributed statistical analyses provide a promising approach for privacy protection when analyzing data distributed over several databases. This approach brings the analysis to the data, rather than the data to the analysis. Instead of directly operating on data, the analyst receives anonymous summary statistics, which are combined into an aggregated result. Further, in model development, it is key to evaluate a trained model w.r.t. to its prognostic or predictive performance. For binary classification, one technique is analyzing the receiver operating characteristics (ROC). Hence, we are interested to calculate the area under the curve (AUC) and ROC curve for a binary classification task using a distributed and privacy-preserving approach.

Methods: We employ DataSHIELD as the technology to carry out distributed analyses, and we use a newly developed algorithm to validate the prediction score by conducting distributed and privacy-preserving ROC analysis. Calibration curves are constructed from mean values over sites. The determination of ROC and its AUC is based on a generalized linear model (GLM) approximation of the ROC curve, the ROC-GLM, as well as on ideas of differential privacy (DP). DP adds noise (quantified by the ℓ_2 sensitivity $\Delta_2(\hat{f})$) to the data. The appropriate choice of the ℓ_2 sensitivity was studied by simulations.

Results: In our simulation scenario, the true and distributed AUC measures differ by $\Delta\text{AUC} < 0.01$ depending on the choice of the differential privacy parameters. It is recommended to check the accuracy of the distributed AUC estimator in specific simulation scenarios when $\Delta_2(\hat{f}) > 0.07$. Here, the accuracy of the distributed AUC estimator may be impaired by too much artificial noise added from DP.

Conclusions: The applicability of our algorithms depends on the sensitivity of the underlying statistical/predictive model. The simulations carried out have shown that the approximation error is acceptable for the majority of simulated cases. For models with high sensitivity, the privacy parameters must be set accordingly higher to ensure sufficient privacy protection, which affects the approximation error. This work shows that complex measures, as the AUC, are applicable for validation in distributed setups while preserving an individual's privacy.

Keywords: Area under the ROC curve; Distributed computing; Medical tests; ROC-GLM

1 Introduction

Medical research is based on the trust that the analysis of confidential patient data follows principles of privacy protection. However, depending on the released data and proposed diagnosis, breaches of the patient's privacy may occur [15]. Even when a patient gives informed consent that the researcher can have access to their pseudonymized patient data, it is necessary to keep data in a protected environment and to process it accordingly. As described by Arellano et al. [1], when the protection of sensitive patient data is a key objective, privacy-preserving modelling should be considered. Typically, multi-centre studies in medicine or epidemiology collect the data in a central study database and perform the analyses in a specifically protected environment following the informed consent of the study subjects. Analogously, in big-data real-world applications, the data of interest may be provided by different locations and may be transferred from there to a central database for analysis. However, this requires an administratively challenging and time-consuming trustworthy data-sharing process.

Using only anonymous and aggregated data for analysis can alleviate the administrative load for data sharing. By reducing the administrative work of conventional data sharing, the new concept of employing distributed data networks in clinical studies makes it possible to leverage routinely collected electronic health data and thus streamline data collection. Non-disclosing distributed analysis is an important part of this concept, as this approach enables statistical analyses without sharing individual patient data (IPD) between the various sites of a clinical study or sharing IPD with a central analysis unit. Thus, non-disclosing distributed analyses protect patient data privacy and enhance data security, making this a potentially advantageous approach for medical research involving sensitive patient data. However, innovative methods are needed to support robust multivariable-adjusted statistical analysis without the need to centralize IPD, thereby providing better protection for patient privacy and confidentiality in multi-database studies.

As a part of the German Medical Informatics Initiative[1] (MII) the Data Integration for Future Medicine (DIFUTURE) consortium [21] undertakes distributed data network studies and provides tools as well as algorithms for non-disclosing distributed analyses. DIFUTURE's specific objective is to provide digital tools for individual treatment decisions and prognosis. Therefore, the development of distributed algorithms for the discovery and validation of prognostic and predictive rules is highly relevant for this mission. In the following paper, we investigate how the area under the curve (AUC) confidence intervals (CIs) proposed by DeLong et al. [5] behave if the computed AUC uses a generalized linear model (GLM) approach of Pepe [19] in a distributed differential privacy framework.

The concept of differential privacy was operationalized by Dwork [6]. An algorithm is considered to be differentially private if an observer cannot determine based solely on the output whether a particular individual's information was used in the computation. Differential privacy is often discussed in the context of ensuring protection of patient data privacy, as differentially private algorithms are more likely to resist identification and reidentification attacks [8] than alternative approaches.

One of the state-of-the-art of prognostic/predictive validation techniques in a binary classification setting is to calculate the receiver operator characteristic (ROC)

[1]www.medizininformatik-initiative.de

curve and its AUC in the pooled data as well as assess the quality of calibration [26]. In general, IPD transfer requires specific patient consent, and data protection laws apply. Here, we present a non-disclosing distributed ROC-GLM, which we use to calculate the ROC curve, its AUC, and the respective CIs. These methods and their implementation in DataSHIELD framework [10] allow analyses in which IPD does not leave its secured environment. This way, only noisy IPD under differential privacy or anonymous and aggregated statistics are shared, thereby preventing the identification of individuals. We also demonstrate that assessing the calibration of binary classification rules is a straightforward task. Thus, non-disclosing distributed validation of prediction and prognostic tools is a milestone in data-driven medicine and can unlock a plethora of medical information for research.

Contribution The work herein provides new privacy-preserving algorithms adapted to the distributed data setting for the ROC-GLM [18], the AUC derived therefrom, and its CIs for that AUC. To validate the algorithms, we provide a simulation study to assess estimation accuracy and to compare the results to the standard approach. Furthermore, we apply the proposed algorithms to validate a given prognostic rule on data of breast cancer patients.

We describe how the adjustments of the distributed ROC analysis are incorporated into (1) the ROC-GLM by using differential privacy [6] to obtain a privacy-preserving survivor function that can be communicated without the threat of privacy breaches and (2) secure aggregations to conduct a distributed Fisher scoring algorithm [13] to obtain parameter estimates for the ROC-GLM. In addition to the ROC analysis to assess the discrimination of a classifier, we describe a distributed calibration approach that respects the privacy of the individuals. Furthermore, we introduce a distributed version of the Brier score [4] and the calibration curve [27].

2 Related literature

Boyd et al. [2] calculate the AUC under differential privacy using a symmetric binormal ROC function. However, our approach is more general, with a possible extension to multiple covariates. While they derive the AUC from the ROC parameters, we use integration techniques. We also provide CIs for the AUC. Ünal et al. [25] use homomorphic encryption to calculate the ROC curve. Their approach does not provide CIs or an extension to multiple covariates. To the best of our knowledge, a modified ROC-GLM algorithm for non-disclosing distributed analyses has so far not been developed.

3 Background

Throughout this paper, we consider binary classification, with 1 for a case with the trait(s) of interest (i.e., “diseased”, “success”, “favorable”) and 0 for the remaining cases (i.e., lacking trait(s) of interest, “healthy”, “no success”, “unfavorable”). Furthermore, $f(\boldsymbol{x}) \in \mathbb{R}$ is the true score based on a true but unknown function f for a patient with a feature vector $\boldsymbol{x}$ of an underlying random vector $\boldsymbol{X}$. In this paper, this score can also express a posterior probability with $f(\boldsymbol{x}) \in [0, 1]$ and is explicitly noted in the corresponding text passages. The function f is estimated by a statistical (classification) model $\hat{f} : \mathbb{R}^p \to \mathbb{R}$. The estimated individual score for a subject with feature or covariate vector $\boldsymbol{x} \in \mathbb{R}^p$ is $\hat{f}(\boldsymbol{x})$. The training data set used to fit $\hat{f}$ is denoted as $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$ with $y_i \in \{1, 0\}$. The score $\hat{f}(\boldsymbol{x})$ and a threshold value $c \in \mathbb{R}$ are used to build a binary classifier: $\mathbb{1}_{[c,\infty)}(\hat{f}(\boldsymbol{x}))$. On an observational level, $\boldsymbol{x}_{1,i}$ and $\boldsymbol{x}_{0,i}$ indicate the i^{th} observation that corresponds to a positive or negative output y. The number of observations in $\mathcal{D}$ with output 1 and 0 are denoted by n_1 and n_0. The set of scores that corresponds to the positive or negative output is denoted by $\mathcal{F}_1 = \{\hat{f}(\boldsymbol{x}_{1,i}) \mid i = 1, \dots, n_1\}$ and $\mathcal{F}_0 = \{\hat{f}(\boldsymbol{x}_{0,i}) \mid i = 1, \dots, n_0\}$, with $\mathcal{F}_{1,i} = \hat{f}(\boldsymbol{x}_{1,i})$ and $\mathcal{F}_{0,i} = \hat{f}(\boldsymbol{x}_{0,i})$.

3.1 ROC curve and AUC

To quantify the quality of a binary classifier, we use the true positive rate (TPR) and false positive rate (FPR) with values between 0 and 1: $\mathsf{TPR}(c) = P(f(\boldsymbol{X}) \geq c \mid Y = 1)$ and $\mathsf{FPR}(c) = P(f(\boldsymbol{X}) \geq c \mid Y = 0)$ for threshold $c \in \mathbb{R}$ [18]. These probability functions are also known as *survivor functions* $S_1(c) = \mathsf{TPR}(c)$ and $S_0(c) = \mathsf{FPR}(c)$. The ROC curve is defined as $\mathrm{ROC}(t) = S_1(S_0^{-1}(t))$. The AUC as a measure of discrimination between the two distributions of the positive and negative class is given as $AUC = \int_0^1 \mathrm{ROC}(t)\ dt$ [30].

3.2 Empirical calculation of the ROC curve and AUC

The calculation of the empirical ROC curve uses the *empirical survivor functions* $\hat{S}_1$ and $\hat{S}_0$. These functions are based on the empirical cumulative distribution functions (ECDF) $\hat{F}_1$ of $\mathcal{F}_1$ and $\hat{F}_0$ of $\mathcal{F}_0$: $\hat{S}_1 = 1 - \hat{F}_1$ and $\hat{S}_0 = 1 - \hat{F}_0$. The set of possible values of the empirical TPR and FPR are given by $\mathcal{S}_1 = \{\hat{S}_1(\hat{f}(\boldsymbol{x}_{0,i})) \mid i = 1, \dots, n_0\}$ and $\mathcal{S}_0 = \{\hat{S}_0(\hat{f}(\boldsymbol{x}_{1,i})) \mid i = 1, \dots, n_1\}$ and are also called *placement values.* These values standardize a given score relative to the class distribution [19]. The set $\mathcal{S}_1$ represents the positive placement values and $\mathcal{S}_0$ the negative placement values.

The empirical version of the $\mathrm{ROC}(t)$ is a discrete function derived from the placement values $\mathcal{S}_1 \subseteq \{0, 1/n_1, \dots, (n_1-1)/n_1, 1\}$ and $\mathcal{S}_0 \subseteq \{0, 1/n_0, \dots, (n_0-1)/n_0, 1\}$. The empirical AUC is then a sum over rectangles of width $1/n_0$ and height $\hat{S}_1(\hat{f}(\boldsymbol{x}_{0,i}))$:

$$\widehat{AUC} = n_0^{-1} \sum_{i=1}^{n_0} \hat{S}_1(\hat{f}(\boldsymbol{x}_{0,i})). \tag{1}$$

3.3 CI for the empirical AUC

The approach proposed by [5] is used to calculate CIs. Here, the variability of the estimated AUC from the empirical variance ($\widehat{\mathsf{var}}$) of the placement values is determined by:

$$\widehat{\mathsf{var}}(AUC) = \frac{\widehat{\mathsf{var}}(\mathcal{S}_1)}{n_0} + \frac{\widehat{\mathsf{var}}(\mathcal{S}_0)}{n_1}. \tag{2}$$

This approach provides a CI for the logit AUC, from which the CI for the AUC can be derived by the logit^{-1} transformation:

$$\mathsf{ci}_\alpha\left(\mathsf{logit}\left(AUC\right)\right) = \mathsf{logit}\left(\widehat{AUC}\right) \pm \Phi^{-1}\left(1-\frac{\alpha}{2}\right)\frac{\sqrt{\widehat{\mathsf{var}}\left(AUC\right)}}{\widehat{AUC}\left(1-\widehat{AUC}\right)}. \tag{3}$$

The term Φ^{-1} denotes the quantile function of the standard normal distribution. Furthermore, statistical testing can be conducted based on that CI. For example, the hypothesis $H_0 : AUC \leq a_0$ vs. $H_1 : AUC > a_0$ with a significance level of α can be tested by checking whether $\mathsf{logit}\left(a_0\right) > a, \forall a \in \mathsf{ci}_\alpha$ to reject H_0.

3.4 The ROC-GLM

The ROC-GLM interprets the ROC curve as a GLM [19, Section 5.5.2]: $\mathrm{ROC}_g(t|\boldsymbol{\gamma}) = g(\boldsymbol{\gamma}h(t))$, with link function $g : \mathbb{R} \to [0,1], \eta \mapsto g(\eta)$, coefficient vector $\boldsymbol{\gamma} \in \mathbb{R}^l$, and covariate vector $h : \mathbb{R} \to \mathbb{R}^l, t \mapsto \boldsymbol{h}(t) = (h_1(t), \ldots, h_l(t))^\mathsf{T}$. The ROC-GLM is an unbiased estimator of the ROC [18]. Estimating the ROC-GLM is based on an intermediate data set $\mathcal{D}_{\text{ROC-GLM}} = \{(u_{ij}, \boldsymbol{h}(t_j)) \mid i = 1, \ldots, n_1, j = 1, \ldots, n_T\}$ with covariates $\boldsymbol{h}(t_j)$, set of thresholds $T = \{t_1, \ldots, t_{n_T}\}$, and binary response $u_{ij} \in \{0,1\}$, $u_{ij} = \mathbb{1}_{(\hat{S}_0(\mathcal{F}_{1,i}),\infty)}(t_j) = \mathbb{1}_{(-\infty,\mathcal{F}_{1,i}]}(\hat{S}_0^{-1}(t_j))$. The simplest ROC-GLM uses the two-dimensional vector $\boldsymbol{h}(t)$ with $h_1(t) = 1$ and $h_2(t) = \Phi^{-1}(t)$. Setting the link function to $g = \Phi$ results in the binormal form $\mathrm{ROC}_g(t|\boldsymbol{\gamma}) = \Phi(\gamma_1 + \gamma_2\Phi^{-1}(t))$ and is represented as a probit regression with response variable u_{ij} and covariate $\Phi^{-1}(t_j)$. A common strategy for choosing the set of thresholds T is to use an equidistant grid.

The estimated ROC curve $\mathrm{ROC}_g(t|\hat{\boldsymbol{\gamma}})$ results from estimating the model parameters $\boldsymbol{\gamma}$ as $\hat{\boldsymbol{\gamma}}$. The estimated AUC from the ROC-GLM $\widehat{AUC}_{\text{ROC-GLM}}$ is obtained by calculating the integral $\widehat{AUC}_{\text{ROC-GLM}} = \int_0^1 \mathrm{ROC}_g(t|\hat{\boldsymbol{\gamma}})\ dt$. Here, we use Rs `integrate` function [20]. Figure 1 visualizes the ROC-GLM algorithm with all individual parts.

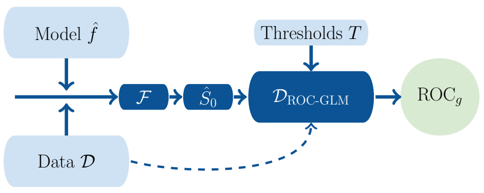

Figure 1 All parts of the $\mathrm{ROCGLM}(\mathcal{D})$ procedure, starting with the data and a model for predicting scores Y, calculating the survivor function $\hat{S}_{\bar{D}}$, and finally calculating intermediate data $\mathcal{D}_{\text{ROC-GLM}}$ for the probit regression to obtain the parameters of the ROC-GLM ROC_g.

3.5 Differential privacy

Following Dwork et al. [9], we add normally distributed noise $\boldsymbol{r}$ to a randomized mechanism $\mathcal{M} : \mathcal{X} \mapsto \mathcal{Y}$ with domain $\mathcal{X}$ (e.g., $\mathcal{X} = \mathbb{R}^p$) and target domain $\mathcal{Y}$

(e.g., $\mathcal{Y} = \mathbb{R}$ in regression) to ensure (ε, δ)-differential privacy [7]. (ε, δ)-differential privacy is given if, for any subset of outputs $R \subseteq \mathcal{Y}$, the property $P(\mathcal{M}(\boldsymbol{x}) \in R) \leq \exp(\varepsilon) P(\mathcal{M}(\boldsymbol{x}') \in R) + \delta$ holds for two adjacent inputs[2] $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$. The value of ε controls how much privacy is guaranteed. The value of δ is the probability that $(\varepsilon, 0)$-differential privacy is broken (also known as ε-differential privacy and the original definition proposed in [8]).

Our randomized mechanism is $\mathcal{M}(\boldsymbol{x}) = \hat{f}(\boldsymbol{x}) + r$. Hence, the protected values of the survivor function are $\widetilde{\mathcal{F}}_1 = \{\mathcal{M}(\boldsymbol{x}_{1,i}) \mid i = 1, \dots, n_1\}$ and not the original score values $\mathcal{F}_1$. The noise $\boldsymbol{r}$ follows a normal distribution $\mathcal{N}(0, \tau^2)$. The variance is set to any value $\tau \geq c\Delta_2(\hat{f})/\varepsilon$ with $c^2 > 2\ln(1.25/\delta)$, $\varepsilon \in (0,1)$, and $\Delta_2(\hat{f})$ is the ℓ_2-sensitivity of $\hat{f}$ measured as $\Delta_2(\hat{f}) = \max_{\text{adjacent } \boldsymbol{x},\, \boldsymbol{x}'} \|\hat{f}(\boldsymbol{x}) - \hat{f}(\boldsymbol{x}')\|_2$. In practice, we first calculate the ℓ_2-sensitivity of the prediction model $\hat{f}$ to determine possible values for ε and δ (see Section 5.3.1). Then, we control the amount of noise added to the algorithm by choosing ε and δ, which sets the variance of the generated noise via $\tau = c\Delta_2(\hat{f})/\varepsilon$ and $c = \sqrt{2\ln(1.25/\delta)}$. Appendix A.2. contains further details and a visualization of the Gaussian mechanism.

4 Distributed ROC-GLM

4.1 General principles

A total of K data sets are distributed over a network of K sites: $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(K)}$. Each data set $\mathcal{D}^{(k)}$ consists of $n^{(k)}$ observations $(\boldsymbol{x}_i^{(k)}, y_i^{(k)})$. The i^{th} feature vector of the k^{th} site is denoted by $x_{\cdot,i}^{(k)}$. The i^{th} outcome on site k is $y_i^{(k)}$. We assume the distributed data to be part of the full but inaccessible data set:

$$\mathcal{D} = \bigcup_{k=1}^{K} \mathcal{D}^{(k)}, \quad n = n^{(1)} + \dots + n^{(K)} \tag{4}$$

Instead of calculating the ROC-GLM for one local data set, we want to calculate the ROC-GLM on K confidential distributed data sets $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(K)}$. All shared information must comply with the following non-disclosing principles:

A1 Aggregated values from which it is not possible to derive original values are shared. Therefore, an aggregation $a : \mathbb{R}^d \mapsto \mathbb{R}$, $\boldsymbol{v} \to a(\boldsymbol{v})$ with $d \geq q \in \mathbb{N}$ must be applied to allow sharing the value $a(\boldsymbol{v})$. The value of q is a *privacy level* guaranteeing that at least q values were used to gain $a(\boldsymbol{v})$. In the distributed setup, the aggregation $a(\boldsymbol{v}^{(k)})$ with $n^{(k)}$ unique values in $\boldsymbol{v}^{(k)}$ shared from each of the K sites can then be further processed. Values $a(\boldsymbol{v}^{(k)})$ are just allowed to be shared if $n^{(k)} \geq q$.

A2 Differential privacy [6] is used to ensure non-disclosive IPD via a noisy representation.

[2] In theory, multiple ways exist to define adjacent inputs. Throughout this article, adjacent inputs are based on a histogram representation $\widetilde{\boldsymbol{x}} \in \mathbb{N}^p$ and $\widetilde{\boldsymbol{x}}' \in \mathbb{N}^p$ of two input vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$. The definition of adjacent inputs is then given by an equal ℓ_1 norm of $\widetilde{\boldsymbol{x}}$ and $\widetilde{\boldsymbol{x}}'$ to one: adjacent $\boldsymbol{x}, \boldsymbol{x}' \Leftrightarrow \|\widetilde{\boldsymbol{x}} - \widetilde{\boldsymbol{x}}'\|_1 = 1$ [cf., 9].

Example: Distributed Brier score and calibration curve Probabilistic (or scoring) classifiers can be assessed by quantifying discrimination and calibration. While the AUC measures discrimination, calibration is often addressed by the Brier score [4] or a calibration curve [27]. Both can be calculated by considering criterion **A1**.

Brier score: The Brier score BS is defined as the mean squared error of the true 0-1-labels and the predicted probabilities of belonging to class 1. For the Brier score, the score $\hat{f}(\boldsymbol{x}) \in [0,1]$ is given as posterior probability. The Brier score is calculated by:

$$\mathrm{BS} = n^{-1} \sum_{i=1}^{n} \left(y_i - \hat{f}(\boldsymbol{x}_i) \right)^2 \tag{5}$$

Hence, having a prediction model $\hat{f}$ at each of the K sites, we can calculate the Brier score by:

1 Calculating the residuals $e_i^{(k)}$ based on the true label $y_i^{(k)}$ at site k and the predicted probabilities $\hat{f}(\boldsymbol{x}_i^{(k)})$: $e_i^{(k)} = y_i^{(k)} - \hat{f}(\boldsymbol{x}_i^{(k)})$, $\forall i = 1, \dots, n^{(k)}$.
2 Calculating $a_{\mathrm{sum}}(\boldsymbol{e}^{(k)} \circ \boldsymbol{e}^{(k)})$, with $\boldsymbol{e}^{(k)} = (e_1^{(k)}, \dots, e_{n^{(k)}}^{(k)})^{\mathsf{T}} \in \mathbb{R}^{n_k}$, the element-wise product $\circ$, and aggregation $a_{\mathrm{sum}}(\boldsymbol{v}^{(k)}) = \sum_{i=1}^{n^{(k)}} v_i^{(k)}$.
3 Sending $a_{\mathrm{sum}}(\boldsymbol{e}^{(k)} \circ \boldsymbol{e}^{(k)})$ and $n^{(k)}$ (if $n_k \geq q$) to the host, who finally calculates $\mathrm{BS} = n^{-1} \sum_{k=1}^{K} a_{\mathrm{sum}}(\boldsymbol{e}^{(k)} \circ \boldsymbol{e}^{(k)})$.

Calibration curve: To calculate a calibration curve, we discretize the domain of the probabilistic classifier $\hat{f}$ in $[0,1]$ into n_{bin} bins (for example, $n_{\mathrm{bin}} + 1$ equidistant points p_i from 0 to 1 to construct the n_{bin} bins $b_l = [p_l, p_{l+1})$ for $l = 1, \dots, n_{\mathrm{bin}} - 1$ and $b_{n_{\mathrm{bin}}} = [p_{n_{\mathrm{bin}}}, p_{n_{\mathrm{bin}}+1}]$ for $l = n_{\mathrm{bin}}$). The calibration curve is the set of 2-dimensional points $p_{\mathrm{cal},l} = (\mathrm{pf}_l, \mathrm{tf}_l)$, with $\mathrm{tf}_l = |\mathcal{I}_l|^{-1} \sum_{i \in \mathcal{I}_l} y_i$ as the true fraction of $y_i = 1$ in bin b_l and $\mathrm{pf}_l = |\mathcal{I}_l|^{-1} \sum_{i \mathcal{I}_l} \hat{f}(\boldsymbol{x}_j)$ as the predicted fraction for outcome 1 in b_l. The set $\mathcal{I}_l$ describes the observations for which the prediction $\hat{f}(\boldsymbol{x}_i)$ falls into bin b_l: $\mathcal{I}_l = \{i \in \{1, \dots, n\} \mid \hat{f}(\boldsymbol{x}_i) \in b_l\}$. A probabilistic classifier $\hat{f}$ is well-calibrated if the points $p_{\mathrm{cal},l}$ are close to the bisector.

In the distributed setup, the points $p_{\mathrm{cal},l}$ are constructed by applying the distributed mean to both points for each bin at each site:

1 Set all $b_1, \dots, b_{n_{\mathrm{bin}}}$, and communicate them to the sites.
2 Calculate the values $c_{l,\mathrm{pf}}^{(k)} = a_{\mathrm{sum}}(\{\hat{f}(\boldsymbol{x}_i^{(k)}) \mid i \in \mathcal{I}_l^{(k)}\})$ and $c_{l,\mathrm{tf}}^{(k)} = a_{\mathrm{sum}}(\{y_i^{(k)} \mid i \in \mathcal{I}_l^{(k)}\})$ for all $l = 1, \dots, n_{\mathrm{bin}}$.
3 Send $\{(c_{l,\mathrm{tf}}^{(k)}, c_{l,\mathrm{pf}}^{(k)}, |\mathcal{I}_l^{(k)}|) \mid k = 1, \dots, K, l = 1, \dots, n_{\mathrm{bin}}\}$ to the host if $|\mathcal{I}_l^{(k)}| \geq q$.
4 The host calculates the calibration curve $p_{\mathrm{cal},l}$ by aggregating the elements $\mathrm{tf}_l = (\sum_{k=1}^{K} |\mathcal{I}_l^{(k)}|)^{-1} \sum_{k=1}^{K} c_{l,\mathrm{tf}}^{(k)}$ and $\mathrm{pf}_l = (\sum_{k=1}^{K} |\mathcal{I}_l^{(k)}|)^{-1} \sum_{k=1}^{K} c_{l,\mathrm{pf}}^{(k)}$ for $l = 1, \dots, n_{\mathrm{bin}}$.

Parts of the distributed ROC-GLM Two aspects are important to construct the distributed version of the ROC-GLM: distrROCGLM. First, the distributed version of the empirical survivor function; Second, a distributed version of the probit regression. Figure 2 shows details of the general procedure. The starting point of the distributed ROC-GLM is the private data $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(K)}$ on the K sites.

The global survivor function $\hat{S}_0$ is approximated by $\widetilde{S}_0$ (Section 4.2) using principle **A2**. The computation of $\widetilde{S}_0$ depends on the level of privacy induced by the (ε, δ)-differential privacy parameters (Section 3.5). The accuracy of the AUC as well as its CI depends on the choice of ε and δ. The global survivor function $\widetilde{S}_0$ is transmitted to each of the K sites and allows calculation of a local version of the intermediate data set $\mathcal{D}^{(k)}_{\text{ROC-GLM}}$ (See Section 3.4). The distributed probit regression complies with principle **A1** and produces the distributed ROC-GLM parameter estimates (see Section 4.3). Using the ROC-GLM of these parameters, denoted by $\widetilde{\text{ROC}}_g$, allows calculation of the approximated AUC, denoted by $\widetilde{AUC}_{\text{ROC-GLM}} = \int_0^1 \widetilde{\text{ROC}}_g(t|\hat{\gamma})\,dt$. Finally, the CIs can be calculated based on a variance estimation, which also complies with principle **A1** (See Section 4.4).

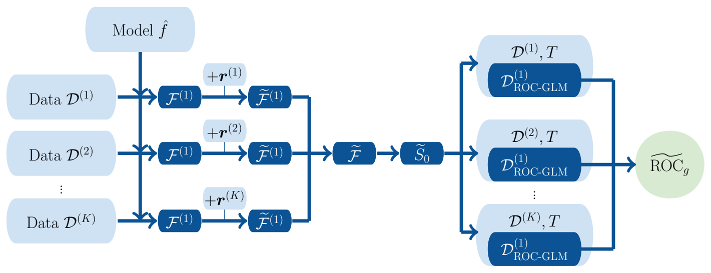

Figure 2 All parts of the distrROCGLM procedure calculating the distributed approximation $\widetilde{\text{ROC}}_g$ of ROC_g. The starting points are the sites (here $K = 3$), which communicate scores with added noise, calculate the global survivor function $\widetilde{S}_{\bar{D}}$, and finally calculate the distributed probit regression on intermediate data $\mathcal{D}^{(k)}_{\text{ROC-GLM}}$ at each site.

4.2 Approximating the global survivor functions

The greatest challenge here is the privacy-preserving calculation of the global survivor function. It is prohibited to directly communicate score values $\mathcal{F}_1^{(k)}$ from the local sites to the analyst. Instead, we propose to calculate an approximation $\widetilde{S}_1$: First, we set the value of ε and τ and generate a noisy representation $\widetilde{\mathcal{F}}_1^{(k)} = \mathcal{F}_1^{(k)} + \boldsymbol{r}^{(k)}$ of the original score values $\mathcal{F}_1^{(k)}$ at each site. Second, the noisy scores are communicated to the host and pooled to $\widetilde{\mathcal{F}}_1 = \bigcup_{k=1}^K \widetilde{\mathcal{F}}_1^{(k)}$ to calculate an approximation $\widetilde{S}_1$ of the global survivor function. Third, (ε, δ)-differential privacy allows sharing $\widetilde{S}_1$ with all sites. Forth, the local sites calculate the global placement values and create the intermediate data set to enter the distributed probit regression.

4.3 Distributed GLM

Existing solutions for distributed computing – such as federated learning [17] – are based on an iterative process of sharing and aggregating parameter values. Although this approach could also be applied to GLMs, it may lead to inexact estimates for heterogeneous data situations [29]. For distributed calculation of the GLM, we use an approach described by [13] and adjust the optimization algorithm of GLMs – the Fisher scoring – at its base to estimate parameters without performance loss. This approach complies with **A1**.

The basis of the ROC-GLM is a probit regression (and therefore a GLM) with $\mathbb{E}(Y \mid X = x) = g(x^{\mathsf{T}}\theta)$ with link function g, response variable Y, and covariates X. The Fisher scoring is an iterative descending technique $\hat{\theta}_{m+1} = \hat{\theta}_m + \mathcal{I}^{-1}(\hat{\theta}_m)\mathcal{V}(\hat{\theta}_m)$ that uses second order gradient information. The components are the score vector $\mathcal{V}(\hat{\theta}_m) = [\partial\ell_\theta(y,x)/\partial\theta]_{\theta=\hat{\theta}_m} \in \mathbb{R}^p$ and the observed Fisher information $\mathcal{I}(\hat{\theta}_m) = [\partial\mathcal{V}(\theta)/\partial\theta]_{\theta=\hat{\theta}_m} \in \mathbb{R}^{p\times p}$ based on the log likelihood $\ell_\theta(\mathcal{D}) = \sum_{i=1}^n \log(f_Y(y_i, x_i))$. A common stop criterion (as used in `R`s [22] `glm` function) to determine whether the Fisher scoring has converged or not is when the relative improvement $|dev_m - dev_{m-1}|/(|dev_m| + 0.1)$ of the deviance $dev_m = -2\,\mathsf{ln}(\ell_{\hat{\theta}_m}(\mathcal{D}))$ is smaller than a value a. The default value used in the `glm` function of `R` is $a = 10^{-8}$.

With non-overlapping data at the K sites (each subject contributes information only at a unique site), condition (4) is fulfilled. This implies the additive structure of the global score vector $\mathcal{V}(\theta_m)$ and Fisher information $\mathcal{I}(\theta_m)$. With the site-specific score vector $\mathcal{V}_k(\theta_m)$ and Fisher information $\mathcal{I}_k(\theta_m)$, it holds:

$$\mathcal{V}(\hat{\theta}_m) = \sum_{k=1}^K \mathcal{V}_k(\hat{\theta}_m) \tag{6}$$

$$\mathcal{I}(\hat{\theta}_m) = \sum_{k=1}^K \mathcal{I}_k(\hat{\theta}_m) \tag{7}$$

This process complies with **A1** and allows estimation of the parameter vector $\hat{\theta}$ with the same precision as in an analysis based on data aggregated over the sites.

4.4 Distributed CIs for the AUC

A straightforward consequence from Section 4.1, is that the distributed calculation of the global sample mean (distrAVG$(\boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(K)})$) complies with **A1**. Here, we provide a distributed version of the sample variance $\widehat{\mathsf{var}}(\boldsymbol{v}) = (n-1)^{-1}\sum_{i=1}^n (v_i - \bar{v})^2$ by a two-step procedure. In the first step, the sample mean is calculated using $\bar{v} = \text{distrAVG}(\boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(K)})$ and shared with all K sites. In the second step, each site calculates the aggregation $a_{\text{var}}(\boldsymbol{v}^{(k)}) = \sum_{i=1}^{n^{(k)}} (v_i^{(k)} - \bar{v})^2$, which is further aggregated to the sample variance $\widehat{\mathsf{var}}(\boldsymbol{v}) = (n-1)^{-1}\sum_{k=1}^K a_{\text{var}}(\boldsymbol{v}^{(k)})$: distrVAR$(\boldsymbol{v}^{(1)}, \dots, \boldsymbol{v}^{(K)})$. The operations distrAVG and distrVAR fulfill **A1** if $n^{(k)} \geq q$, $\forall k \in \{1, \dots, K\}$.

Based on operation distrVAR, non-disclosing distributed CIs for the global AUC can be provided. As described in Section 3.2 and Section 3.3, the calculation of the approximated CI requires both approximated survivor functions $\widetilde{S}_0$ and $\widetilde{S}_1$ (see Section 4.2). A distributed CI $\widetilde{\mathsf{ci}}_\alpha$ to approximate ci_α follows from Formula (3).

5 Simulation study

5.1 General considerations

It is emphasized in Section 4.3 that the survivor functions for the data at hand are needed to build placement values and to create the data set for the probit regression in order to estimate the ROC curve and its AUC. Based on the Gaussian mechanism, noise is generated to create a non-disclosing distributed survivor function. The aim of the simulation study is to understand the effect of the introduced noise (which is necessary to conduct the distributed analysis) on the accuracy when compared to the empirical AUC and the CI of DeLong et al. [5]. We assume that the well-studied empirical AUC [11, 16] and CI are adequate estimators of the true AUC of the underlying data generating process that is already attached with a certain estimation error. Our goal is not to construct better estimates for the true AUC, but to study the difference between our distributed approach to the estimates applied to the pooled data.

In our simulation, we explore the bias introduced by our distributed approach. To assess the accuracy of our distributed approach when estimating the AUC, we measure the difference $\Delta AUC = AUC - \widetilde{AUC}_{\text{ROC-GLM}}$ of the AUC obtained by the distributed ROC-GLM $\widetilde{AUC}_{\text{ROC-GLM}}$ (Section 4.1) and the empirical AUC (Section 3.2). Of interest is obtaining an accuracy of $|\Delta AUC| \leq 0.01$.

For the CI, we calculate the error $\Delta\, \mathsf{ci}_\alpha$ based on the symmetric difference between ci_α proposed by DeLong et al. [5, see Section 3.3] and our non-disclosing distributed approach $\widetilde{\mathsf{ci}}_\alpha$ (Section 4.4). We study $\Delta\, \mathsf{ci}_\alpha = |\widetilde{\mathsf{ci}}_{\alpha,l} - \mathsf{ci}_{\alpha,l}| + |\widetilde{\mathsf{ci}}_{\alpha,r} - \mathsf{ci}_{\alpha,r}|$, with indices l and r denoting the left and right side of the CI, respectively. It is of interest to have an error smaller than 0.01: $\Delta\, \mathsf{ci}_\alpha < 0.01$.

We explore the following research questions:

Question 1 – Correctness of the ROC-GLM and distributed ROC-GLM (Section 5.3.1): How can we set both privacy parameters ε and δ to reach $|\Delta AUC|$ below 0.01?

Question 2 – Correctness of the AUC CIs (Section 5.3.2): How can we set both privacy parameters ε and δ to reach $\Delta\, \mathsf{ci}_\alpha$ below 0.01?

5.2 Data generation

The aim of the following data generation is to simulate uniformly distributed AUC values between 0.5 and 1. (1) The data generation starts with randomly picking n from $\{100, \ldots, 2500\}$. (2) For each $i \in \{1, \ldots, n\}$, the *true* prediction scores are generated from the uniform distribution $\mathcal{F}_i \sim U[0,1]$. Next, (3) the class membership $y_i \in \{0,1\}$ is determined by $y_i = \mathbb{1}(\mathcal{F}_i \geq 0.5)$. This results in a perfect AUC of 1. (4) The perfect ordering of the class values with respect to individual scores is broken by flipping labels randomly. A set of indexes $\mathcal{I}$ of size $\lfloor \gamma n \rfloor$ is selected for which the corresponding labels are replaced by $y_i \sim \text{Ber}(0.5)$, $\forall i \in \mathcal{I}$. The fraction γ is sampled from a $U[0.5; 1]$ distribution. (5) For comparison, the empirical AUC is calculated from the vector of scores $\mathcal{F}$ and flipped labels y. (6) The non-disclosing distributed process described in Section 4.1 is used to calculate the $\widetilde{AUC}_{\text{ROC-GLM}}$

and $\widetilde{ci}_{0.05}$. The examined values for the distributed ROC-GLM are described in Section 5.3.1. The simulation is repeated 10000 times.

To demonstrate the effectiveness of the basic non-distributed ROC-GLM AUC estimation, Figure 3 shows the empirical distribution of the empirical as well as ROC-GLM-based AUC values depending on the sizes of n. The distribution of the empirical AUC values is close to the uniform distribution over the range of 0.5 to 1. The behaviour of the distribution at the borders can be explained as follows: To obtain an AUC value of one, it is necessary to keep all original class labels y. However, this happens rarely, due to the randomized assignment of the observations chosen in $\mathcal{I}$. The same applies to AUC values close to 0.5. An AUC value of 0.5 appears if the class labels are completely randomized. This is also a rare event.

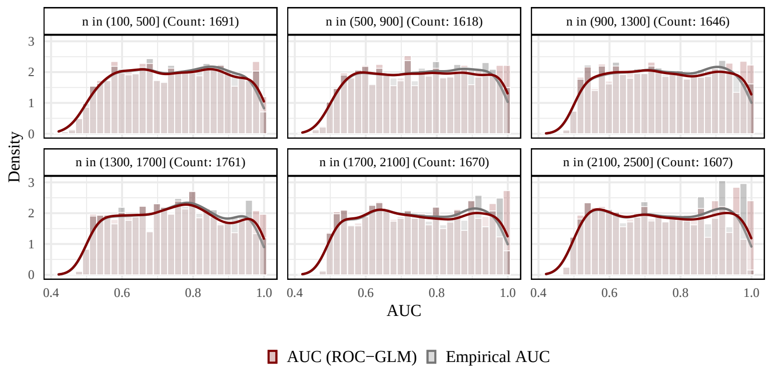

Figure 3 Densities of 10 000 simulated values of the empirical AUC and AUC from the ROC-GLM. The densities are grouped by different data sizes n.

5.3 Results

5.3.1 Correctness of the ROC-GLM and distributed ROC-GLM

ROC-GLM Figure 3 shows a nearly perfect overlap of the empirical distributions of the empirical as well as basic non-distributed ROC-GLM-based AUC values in the range of values between 0.6 and 0.8. Nevertheless, the behaviour at the right border results from the fact that the response U of the probit regression contains only very few values of zero and mostly values of 1, resulting in an unbalanced data situation. This impairs the numerical behaviour of the probit regression estimation.

Next, we quantify the difference between the empirical and the basic non-distributed ROC-GLM-based AUC estimates: $(AUC - AUC_{\text{ROC-GLM}})$. Table 1 shows summary statistics of these differences organized by bins of the empirical AUC of width 0.025. In **Question 1**, an absolute difference below 0.01 is requested, which is fulfilled over the whole AUC range. The mean and median differences for AUC values ranging from 0.5 to 0.95 fulfil this requirement, whereas AUC values between 0.95 and 0.975 show slightly larger differences.

Emp. AUC (Bin)	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Sd.	Count
$(0.5, 0.525]$	-0.0044	-0.0002	0.0002	0.0003	0.0008	0.0053	0.0009	384
$(0.525, 0.55]$	-0.0052	0.0000	0.0006	0.0006	0.0011	0.0042	0.0010	490
$(0.55, 0.575]$	-0.0031	0.0003	0.0009	0.0009	0.0015	0.0052	0.0010	463
$(0.575, 0.6]$	-0.0018	0.0006	0.0012	0.0012	0.0017	0.0052	0.0010	481
$(0.6, 0.625]$	-0.0044	0.0009	0.0015	0.0014	0.0020	0.0064	0.0010	485
$(0.625, 0.65]$	-0.0039	0.0012	0.0017	0.0017	0.0022	0.0069	0.0010	501
$(0.65, 0.675]$	-0.0031	0.0013	0.0018	0.0018	0.0023	0.0068	0.0011	503
$(0.675, 0.7]$	-0.0022	0.0012	0.0018	0.0018	0.0023	0.0064	0.0010	465
$(0.7, 0.725]$	-0.0082	0.0010	0.0016	0.0016	0.0023	0.0070	0.0012	523
$(0.725, 0.75]$	-0.0031	0.0008	0.0015	0.0014	0.0021	0.0087	0.0012	485
$(0.75, 0.775]$	-0.0058	0.0004	0.0011	0.0010	0.0018	0.0053	0.0013	501
$(0.775, 0.8]$	-0.0053	-0.0003	0.0004	0.0005	0.0012	0.0088	0.0015	523
$(0.8, 0.825]$	-0.0061	-0.0013	-0.0002	-0.0004	0.0005	0.0045	0.0016	476
$(0.825, 0.85]$	-0.0125	-0.0023	-0.0013	-0.0014	-0.0003	0.0059	0.0019	484
$(0.85, 0.875]$	-0.0111	-0.0037	-0.0026	-0.0025	-0.0014	0.0074	0.0020	520
$(0.875, 0.9]$	-0.0136	-0.0056	-0.0044	-0.0043	-0.0030	0.0076	0.0023	534
$(0.9, 0.925]$	-0.0195	-0.0080	-0.0065	-0.0065	-0.0052	0.0066	0.0026	515
$(0.925, 0.95]$	-0.0193	-0.0105	-0.0091	-0.0089	-0.0076	0.0056	0.0030	481
$(0.95, 0.975]$	-0.0227	-0.0138	$\mathbf{-0.0113}$	$\mathbf{-0.0113}$	-0.0093	0.0067	0.0037	503
$(0.975, 1]$	-0.0180	-0.0093	-0.0062	-0.0064	-0.0034	0.0013	0.0039	529

Table 1 Minimum, 0.25-quantile/1st quantile, median, mean, 0.75-quantile/3rd quantile, maximum, standard deviation, and the differences $AUC - AUC_{\text{ROC-GLM}}$ of the bins containing the respective subset of the 10000 empirical AUC values. Bold values indicate that these AUC bins are not smaller than 0.01 as demanded by **Question 1**. The count column indicates the number of simulated AUC values per bin.

Distributed ROC-GLM In the following, we investigate the accuracy of the AUC estimated by the distributed ROC-GLM. Differential privacy – a necessary component – is determined by the parameters ε and δ. These parameters must be determined in such a way that **Question 1** holds. The data are distributed over five sites: The simulated prediction scores $\mathcal{F}$ and true classes y are randomly split into $K = 5$ parts $\mathcal{F}^{(1)}, \dots, \mathcal{F}^{(5)}$ and $y^{(1)}, \dots, y^{(5)}$. Our simulation setting uses $\varepsilon \in A_\varepsilon = \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and $\delta \in A_\delta = \{0.1, 0.2, 0.3, 0.4, 0.5\}$. Due to the Gaussian mechanism, we must also take the ℓ_2-sensitivity into account. We assume $\Delta_2(\hat{f}) \in A_{\Delta_2(\hat{f})} = \{0.01, 0.03, 0.05, 0.07, 0.09\}$. For the simulation, each setting of the grid $A_\varepsilon \times A_\delta \times A_{\Delta_2(\hat{f})}$ is evaluated by simulating 10000 data sets (cf. Section 5.2) and hence obtaining 10000 $\widetilde{AUC}_{\text{ROC-GLM}}$ values that are compared to the respective empirical AUC.

Figure 4 shows the simulation results for different ε and δ combinations. The absolute difference of the empirical AUC on the pooled data and the AUC based on the distributed ROC-GLM is checked for having a value below 0.01. The results are based on 10000 simulation runs for 25 $\varepsilon - \delta$-combinations and for each $\Delta_2(\hat{f}) \in \{0.01, 0.030.05, 0.07, 0.09\}$.

Figure 4 reveals that the bias between empirical and distributed AUC depends on the ℓ_2-sensitivity. The smaller the sensitivity and hence the better the model $\hat{f}$, less noise is required to ensure privacy. Correspondingly, smaller choices of privacy parameters can and should be used to ensure privacy. Based on the results, we choose $(\varepsilon, \delta) = (0.2, 0.1)$ for $\Delta_2(\hat{f}) \leq 0.01$, $(\varepsilon, \delta) = (0.3, 0.4)$ for $\Delta_2(\hat{f}) \in (0.01, 0.03]$, $(\varepsilon, \delta) = (0.5, 0.3)$ for $\Delta_2(\hat{f}) \in (0.03, 0.05]$, and $(\varepsilon, \delta) = (0.5, 0.5)$ for $\Delta_2(\hat{f}) \in (0.05, 0.07]$. Based on the simulation, we recommend using our distributed approach for settings with $\Delta_2(\hat{f}) > 0.07$ with caution, and we highlight that the accuracy of the AUC estimator suffers because of too much generated noise.

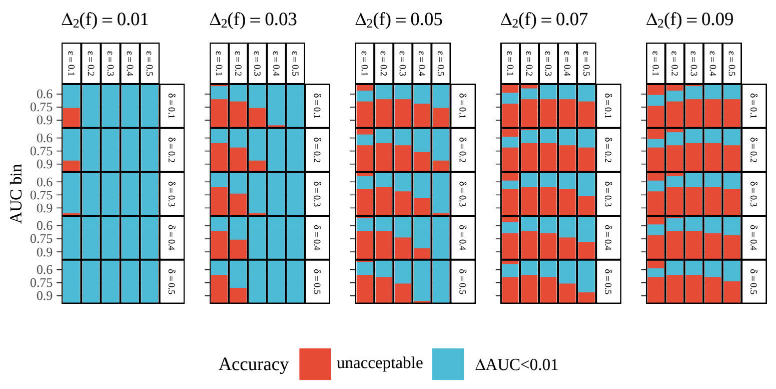

Figure 4 Combinations of the privacy parameters ε and δ and their applicability. Each rectangle contains empirical AUC bins of size 0.025 (cf. Table 1) and visualizes the mean of the absolute difference $|\Delta AUC|$ (mean absolute error, MAE) of the distributed AUC compared to the empirical AUC per bin. Each rectangle corresponds to one simulation setting $(\Delta_2(\hat{f}), \varepsilon, \delta)$. The MAE per bin is categorized according to our hypothesis, with blue visualizing an MAE ≤ 0.01 (**Question 1**) while red shows an unacceptable accuracy measured as MAE larger than 0.01.

5.3.2 Correctness of the AUC CIs

The respective results in terms of acceptable (ε, δ) combinations are shown in Figure 5. Acceptable (ε, δ) combinations under **Question 1** are also acceptable under **Question 2**. Therefore, we recommend using the more restrictive settings described in the previous Section 5.3.1 for the AUC estimation of the distributed ROC-GLM.

6 Data analysis

In this chapter, we develop a prognostic model and validate its predictive performance on a distributed test data set. The following presents the distributed analysis, which is also compared to the pooled analysis (see Section 6.4). As a privacy level, we choose a value of $q = 5$ (see Section 4.1, **A1**).

About the data The data set is provided by the German Breast Cancer Study Group [24] and can be found in the `TH.data` package [12]. The data consists of records from 686 breast cancer patients on the effect of hormonal therapy on survival. Besides the binary variable hormonal treatment (`horTH`), the data set provides information on age (`age`), menopausal status (`menostat`), tumor size (in mm, `tsize`), tumor grade (`tgrade`), number of positive nodes (`pnodes`), progesterone receptor (in fmol, `progrec`), estrogen receptor (in fmol, `estrec`), recurrence-free survival time (in days, `time`), and censoring indicator (0- censored, 1- event, `cens`).

Because the data set is (by its nature) not distributed, we use 60 % (412 observations) for training the model and split the remaining 40 % (274 observations) into 5 parts $\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(5)}$ with $n^{(1)} = 56$, $n^{(2)} = 49$, $n^{(3)} = 60$, $n^{(4)} = 49$, and $n^{(5)} = 60$ that are used for the distributed validation. Each split is distributed to a site to simulate the distributed setup.

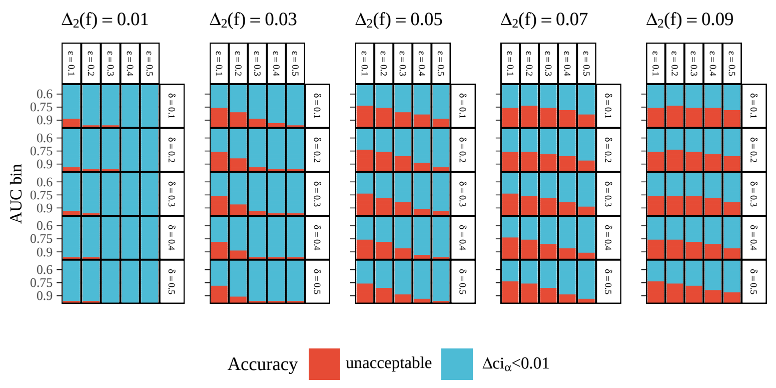

Figure 5 Combinations of the privacy parameters ε and δ and their applicability depending on $\Delta_2(\hat{f})$. Each rectangle contains empirical AUC bins of size 0.025 (cf. Table 1) and visualizes the mean of the relative error $\Delta\,\mathrm{ci}_{0.05}$ of the distributed CI $\widetilde{ci}_{0.05}$ compared to $\mathrm{ci}_{0.05}$. Blue shows accuracy values with $\Delta\,\mathrm{ci}_{0.05} \leq 0.01$ (**Question 2** applies), while red visualizes inaccuracies of $\Delta\,\mathrm{ci} > 0.01$.

The aim is to predict the survival probability $p(t|\boldsymbol{x}) = P(T > t|X = \boldsymbol{x})$ of surviving time point t based on covariates $\boldsymbol{x}$. For the use case, we choose $t = 730$ (two years), and therefore, the goal is to validate the survival probability of a patient after two years in the study. The predicted scores are the survival probabilities $\hat{y}_i = \hat{f}(\boldsymbol{x}_i) = \hat{p}(730|\boldsymbol{x}_i)$ with $\boldsymbol{x}_i \in \cup_{k=1}^{K}\mathcal{D}^{(k)}$. The corresponding binary variable y_i equals 0 if the patient dies in $[0, 730]$ or a recurrence was observed, and y_i equals 1 if otherwise. Therefore, a high value for the survival probability $\hat{y}_i$ ideally corresponds to a binary outcome of 1.

About the model We choose a random forest [3] using the R package `ranger` [28] as a prognostic model $\hat{f}$ for the survival probability $p(t|\boldsymbol{x})$. With the exception of the number of trees (which is set to 20), the random forest was trained with the default hyperparameter settings of the `ranger` implementation. The model formula is given by

$$\mathsf{Surv(time,\ cens)} \sim \mathsf{horTh} + \mathsf{age} + \mathsf{tsize} + \mathsf{tgrade} + \mathsf{pnodes} + \mathsf{progrec} + \mathsf{estrec}.$$

About the implementation The implementation is based on the DataSHIELD [10] framework and is provided by an R package called `dsBinVal` (github.com/difuture-lmu/dsBinVal). Further details about these methods and privacy considerations can be found in the respective GitHub README.

Aim of the analysis The main goal of the analysis is to test the hypothesis that the true AUC is significantly larger than 0.6 as the minimal prognostic performance of the model $\hat{f}$. The significance level is set to $\alpha = 0.05$:

$$H_0: \ AUC \leq 0.6 \ \text{ vs. } \ H_1: \ AUC > 0.6 \qquad (8)$$

To test the hypothesis, we estimate the AUC with $\widetilde{AUC}_{\text{ROC-GLM}}$ using the distributed ROC-GLM as well as the approximated CI $\widetilde{\mathsf{ci}}_{0.05}$. We reject H_0 if $AUC > 0.6,\ \forall AUC \in \widetilde{\mathsf{ci}}_{0.05}$.

Analysis plan In the following, (1) we start in with the calculation of the ℓ_2-sensitivity (Section 6.1). Depending on the result, we set the privacy parameters ε and δ. Next, (2) we continue with fitting the distributed ROC-GLM and calculating the approximation of the AUC CI (Section 6.2). At this point, we are able to make a decision about the hypothesis in equation (8). In a final step, (3) we demonstrate how to check the calibration of the model using the distributed Brier score and calibration curve (Section 6.3).

6.1 Choice of the privacy parameters

Given the model and the data set, the ℓ_2-sensitivity is $\Delta_2(\hat{f}) = 0.016$. Following the results of Section 5.3.1, we use $\varepsilon = 0.3$ and $\delta = 0.4$, as suggested for $\Delta_2(\hat{f}) \in (0.01, 0.03]$.

6.2 Calculation of the distributed ROC-GLM

The fit of the ROC-GLM results in parameter estimates of $\gamma_1 = 0.7817$ and $\gamma_2 = 1.2486$. The AUC obtained from the ROC curve using these parameters is $AUC_{\text{ROC-GLM}} = 0.6875$ with $\widetilde{\mathsf{ci}}_{0.05} = [0.6051, 0.7595]$. The results are visualized in Figure 6.

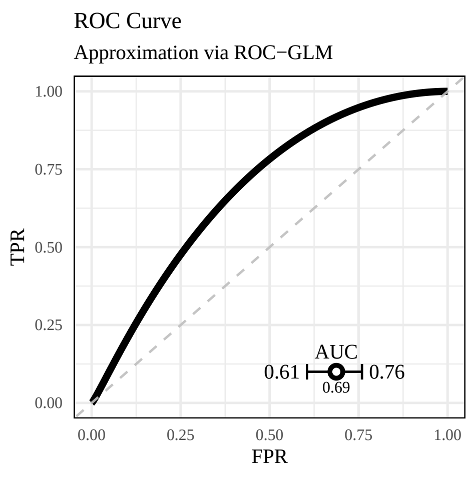

Figure 6 ROC curve estimated by the distributed ROC-GLM.

Based on the given CI, we significantly reject H_0 for H_1 and hence assume the true AUC to be greater than 0.6.

6.3 Checking the calibration

The Brier score of $\hat{f}$ calculates to $\text{BS} = 0.1733$ and indicates a good but not perfect calibration. We further assume our model to be not calibrated perfectly. Still, the calibration is adequate, but the model seems to underestimate the true relative frequencies for scores greater than 0.3. Figure 7 shows the distributed calibration curve as well as the individual calibration curves per site. Furthermore, we observe that the range of the calibration curve does not cover the whole range of the scores $\hat{f}(x) \in [0, 1]$. This indicates that our model does not predict scores close to 1. We want to highlight that, due to privacy reasons, not all score values were included in the calculation; aggregated values are only shared if they consist of at least 5 elements. The table in Appendix A.3 shows the number of elements per bin and site.

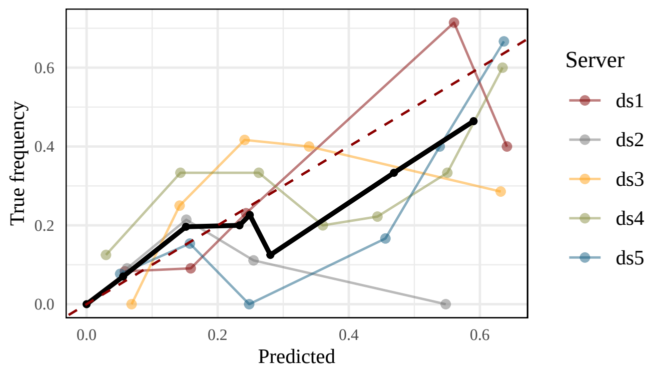

Figure 7 Calibration curve (bold line) and calibration curves of the individual sites using 10 bins. Note that aggregated values from the site are only shared if one bin contains more than 5 values. See Appendix A.3 for tables containing the numbers of values per bin.

6.4 Comparison with pooled data

Comparing the ROC curves using the empirical ROC and the distributed ROC-GLM (Figure 8, left) shows a good fit of the ROC-GLM. The resulting AUC values are $\widetilde{AUC}_{\text{ROC-GLM}} = 0.6875$ and $AUC = 0.6919$ with $|\Delta AUC| = 0.0044 < 0.01$. The CIs of the approximated CI $\widetilde{\mathsf{ci}}_{0.05} = [0.6051, 0.7595]$ and the CI on the pooled scores $\mathsf{ci}_{0.05} = [0.6131, 0.7608]$ reveals a slightly more pessimistic CI estimation in the distributed setup. The error of the CI calculates to $\Delta\,\mathsf{ci}_{0.05} = 0.0094 < 0.01$.

The distributed calibration curve shows a good overlap with the calibration curve in areas where all data are allowed to be shared. For bins where this is not the case, the distributed calibration curve is off. Still, the tendency of over- or underestimation of the distributed calibration curve corresponds to one of the pooled curves. The bins for which the full information was received are $[0, 0.1]$, $(0.1, 0.2]$, and $(0.2, 0.3]$ (cf. Appendix A.3 table 1). For all other bins, at least one site was not allowed to share the aggregated values. The Brier score of the pooled and distributed approach is equal.

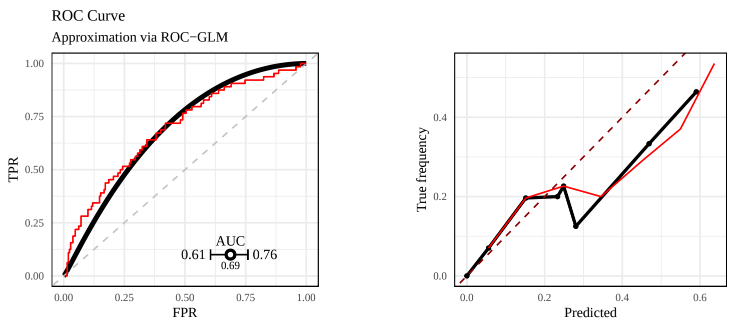

Figure 8 Comparison of the empirical ROC curve with ROC curve obtained by the distributed ROC-GLM (left). Comparison of the calibration curve when calculated on the pooled scores compared with the distributed calibration curve (right). The thin curves are the lines on the pooled data.

7 Reproducibility considerations

All experiments were conducted using `R` version 4.1.2 on a Linux machine with an Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz processor. The package used to run the simulation was `batchtools` [14]. The code to reproduce all results as well as all simulation results is available in a GitHub repository[3]. The repository contains a README file with further details and a script to install all packages with the respective version used when the benchmark was conducted. Furthermore, a Docker image[4] can be installed providing a snapshot of the system at the time of the benchmark containing `R` and all packages with their respective version. The Docker image also comes with the RStudio container[5] that allows direct inspection of all the results in a web browser.

The code to conduct the data analysis is given in a separate GitHub repository[6]. The repository contains the data, an installation of all necessary packages, as well as code to set up the publicly available DataSHIELD server[7] to run the analysis[8].

8 Discussion

Distributed non-disclosing (i.e., privacy-preserving) strategies for data analysis are highly relevant for data-driven biomedical research. Since the analyses can be considered anonymous, current legal data protection frameworks allow their use without requesting specific consent. Protecting privacy by appropriate means is fundamental when using personal data for research. These technologies also enable taking

[3] github.com/difuture-lmu/simulations-distr-auc
[4] hub.docker.com/repository/docker/schalkdaniel/simulations-distr-auc
[5] hub.docker.com/r/rocker/rstudio
[6] github.com/difuture-lmu/datashield-roc-glm-demo
[7] Available at opal-demo.obiba.org. The reference, username, and password are available at the OPAL documentation opaldoc.obiba.org/en/latest/resources.html in the “Types” section.
[8] We cannot guarantee the functionality of the DataSHIELD server or if it will be publicly available forever. However, we keep the repository up-to-date by using continuous integration, which is triggered automatically every week. This system also reports errors that occur if the analysis cannot be conducted on the test server anymore. Further information can be found in the README file of the repository.

part in broader network structures without additional administrative work concerning data protection issues. Privacy-preserving distributed computation allows researchers to digitally cooperate and leverage the value of their data while respecting data sovereignty and without compromising privacy. Besides the privacy preservation in algorithms that are backed up with security mechanisms, it is worth noting that software is also a key player in privacy-preserving analysis. For example, most models fitted with the statistical software `R` attach data directly to the model object. Sharing these objects without caution gives analysts direct access to the training data [cf., e.g., 23].

International activity has been dedicated to setting up distributed non-disclosing analysis frameworks, which implement machine learning approaches into a distributed analysis scheme. The availability of the respective algorithms is growing, and distributed learning for data from heterogeneous clinical servers has emerged as a hot field. However, our impression is that algorithms for distributed validation of these learning algorithms are lacking.

In this paper, we specifically focused on the assessment of discrimination and calibration of learning algorithms with a binary outcome. The discrimination is estimated by a ROC curve and its AUC. We also provide CIs to the distributed AUC estimate. The distributed estimation process is based on *placement values* and *survivor functions*. They represent qualities of the global distribution of score values (aggregated over all centers). To do this in a non-disclosing way, we applied differential privacy techniques. With the creation of the placement values and the transmission of this information to the local server, we applied a distributed version of the ROC-GLM approach to estimate the ROC curve and its AUC in a distributed way. We used a straightforward approach for the distributed GLM estimation. However, we acknowledge that there may be more efficient approaches, and we will explore this aspect in future work.

Abbreviations

AUC: Area under the curve; CI: Confidence interval; DP: Differential privacy; FPR: False positive rate; GLM: Generalized linear model; IPD: Individual patient data; MII: Medical Informatics Initiative; ROC: Receiver operating characteristics; TPR: True positive rate.

Declarations

Ethical Approval and consent to participate

Not applicable.

Consent to Publication

Not applicable.

Data Availability statement

The simulated datasets generated during the current study are available on GitHub, https://github.com/difuture-lmu/simulations-distr-auc.

Conflict of interest

The authors declare no competing interests.

Funding

Not applicable.

Acknowledgment

This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A and Federal Ministry for Research and Technology (BMFT) under Grant No. 01ZZ1804C (DIFUTURE, MII). The authors of this work take full responsibility for its content.

Author contribution

DS wrote the manuscript, implemented the methods and the simulation study, and prepared the use case. DS also created all graphics and the interpretation of the results from the simulation study and from the use case. In addition, much effort was put into reproducibility, for which DS created a Docker container and a GitHub repository with the simulation study results. The idea of the distributed AUC calculation originated from UM. All co-authors provided substantial assistance in writing the manuscript and interpreting the simulation study results.

Author details

[1]*Department of Statistics, LMU Munich, Munich, Germany.* [2]*Institute for Medical Information Processing, Biometry and Epidemiology, LMU Munich, Munich, Germany.* [3]*DIFUTURE (DataIntegration for Future Medicine, www.difuture.de), LMU Munich, Munich, Germany.* [4]*Munich Center for Machine Learning (MCML).*

References

1. *Arellano, A. M., Dai, W., Wang, S., Jiang, X., and Ohno-Machado, L. (2018). Privacy policy and technology in biomedical data science.* Annual review of biomedical data science, *1:115–129.*
2. *Boyd, K., Lantz, E., and Page, D. (2015). Differential privacy for classifier evaluation. In* Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, *pages 15–23.*
3. *Breiman, L. (2001). Random forests.* Machine learning, *45(1):5–32.*
4. *Brier, G. W. et al. (1950). Verification of forecasts expressed in terms of probability.* Monthly weather review, *78(1):1–3.*
5. *DeLong, E. R., DeLong, D. M., and Clarke-Pearson, D. L. (1988). Comparing the areas under two or more correlated receiver operating characteristic curves: a nonparametric approach.* Biometrics, *pages 837–845.*
6. *Dwork, C. (2006). Differential privacy. In* International Colloquium on Automata, Languages, and Programming, *pages 1–12. Springer.*
7. *Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., and Naor, M. (2006a). Our data, ourselves: Privacy via distributed noise generation. In* Annual International Conference on the Theory and Applications of Cryptographic Techniques, *pages 486–503. Springer.*
8. *Dwork, C., McSherry, F., Nissim, K., and Smith, A. (2006b). Calibrating noise to sensitivity in private data analysis. In* Theory of cryptography conference, *pages 265–284. Springer.*
9. *Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy.* Found. Trends Theor. Comput. Sci., *9(3-4):211–407.*
10. *Gaye, A., Marcon, Y., Isaeva, J., LaFlamme, P., Turner, A., Jones, E. M., Minion, J., Boyd, A. W., Newby, C. J., Nuotio, M.-L., et al. (2014). Datashield: taking the analysis to the data, not the data to the analysis.* International journal of epidemiology, *43(6):1929–1944.*
11. *Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (roc) curve.* Radiology, *143(1):29–36.*
12. *Hothorn, T. (2021).* TH.data: TH's Data Archive. *R package version 1.1-0.*
13. *Jones, E. M., Sheehan, N. A., Gaye, A., Laflamme, P., and Burton, P. (2013). Combined analysis of correlated data when data cannot be pooled.* Stat, *2(1):72–85.*
14. *Lang, M., Bischl, B., and Surmann, D. (2017). batchtools: Tools for r to work on batch systems.* The Journal of Open Source Software, *2(10).*
15. *Loukides, G., Denny, J. C., and Malin, B. (2010). The disclosure of diagnosis codes can breach research participants' privacy.* Journal of the American Medical Informatics Association, *17(3):322–327.*
16. *Mason, S. J. and Graham, N. E. (2002). Areas beneath the relative operating characteristics (roc) and relative operating levels (rol) curves: Statistical significance and interpretation.* Quarterly Journal of the Royal Meteorological Society: A journal of the atmospheric sciences, applied meteorology and physical oceanography, *128(584):2145–2166.*
17. *McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In* Artificial intelligence and statistics, *pages 1273–1282. PMLR.*

18. *Pepe, M. S. (2000). An interpretation for the roc curve and inference using glm procedures.* Biometrics, *56(2):352–359.*
19. *Pepe, M. S. (2003). The Statistical Evaluation of Medical Tests for Classification and Prediction.* Journal of the American Statistical Association.
20. *Piessens, R., de Doncker-Kapenga, E., Überhuber, C. W., and Kahaner, D. K. (2012).* Quadpack: a subroutine package for automatic integration, *volume 1. Springer Science & Business Media.*
21. *Prasser, F., Kohlbacher, O., Mansmann, U., Bauer, B., and Kuhn, K. A. (2018). Data integration for future medicine (difuture).* Methods Inf Med, *57(S01):e57–e65.*
22. *R Core Team (2021).* R: A Language and Environment for Statistical Computing. *R Foundation for Statistical Computing, Vienna, Austria.*
23. *Schalk, D., Irmak On, B., Hapfelmeier, A., Mansmann, U., and Hoffmann, V. S. (2022). Model transportability and privacy protection. 31st International Biometric Conference, github.com/schalkdaniel/talk-ibc-2022/blob/main/model-transportability-and-privacy-protection.pdf.*
24. *Schumacher, M., Bastert, G., Bojar, H., Hübner, K., Olschewski, M., Sauerbrei, W., Schmoor, C., Beyerle, C., Neumann, R., and Rauschecker, H. (1994). Randomized 2 x 2 trial evaluating hormonal treatment and the duration of chemotherapy in node-positive breast cancer patients. german breast cancer study group.* Journal of Clinical Oncology, *12(10):2086–2093.*
25. *Ünal, A. B., Pfeifer, N., and Akgün, M. (2021). ppaurora: Privacy preserving area under receiver operating characteristic and precision-recall curves with secure 3-party computation.* ArXiv, *2102.*
26. *Van Calster, B., McLernon, D. J., Van Smeden, M., Wynants, L., and Steyerberg, E. W. (2019). Calibration: the achilles heel of predictive analytics.* BMC medicine, *17(1):1–7.*
27. *Vuk, M. and Curk, T. (2006). Roc curve, lift chart and calibration plot.* Metodoloski zvezki, *3(1):89.*
28. *Wright, M. N. and Ziegler, A. (2017). ranger: A fast implementation of random forests for high dimensional data in C++ and R.* Journal of Statistical Software, *77(1):1–17.*
29. *Yang, C., Wang, Q., Xu, M., Chen, Z., Bian, K., Liu, Y., and Liu, X. (2021). Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. In* Proceedings of the Web Conference 2021, *pages 935–946.*
30. *Zweig, M. H. and Campbell, G. (1993). Receiver-operating characteristic (roc) plots: a fundamental evaluation tool in clinical medicine.* Clinical chemistry, *39(4):561–577.*

CHAPTER 11

`dsBinVal`: Conducting distributed ROC analysis using DataSHIELD

Contributing article

Schalk, D., Hoffmann, V. S., Bischl, B., and Mansmann, U. (2023b). dsBinVal: Conducting distributed roc analysis using datashield. *Journal of Open Source Software*, 8(82):4545

Declaration of contributions

The software was developed exclusively by Daniel Schalk. He also set up automatic unit tests for functionality and code quality for various operating systems (Linux, Windows, macOS). Particular emphasis was placed on data protection concepts and requirements for integration with DataSHIELD. He mainly wrote the manuscript.

Contribution of the coauthors

Verena Hoffmann advised data protection concepts on the implementation. All co-authors helped revise the manuscript.

The Journal of Open Source Software

dsBinVal: Conducting distributed ROC analysis using DataSHIELD

Daniel Schalk[1,3,4], **Verena Sophia Hoffmann**[2,3], **Bernd Bischl**[1,4], and **Ulrich Mansmann**[2,3]

1 Department of Statistics, LMU Munich, Munich, Germany **2** Institute for Medical Information Processing, Biometry and Epidemiology, LMU Munich, Munich, Germany **3** DIFUTURE (DataIntegration for Future Medicine, www.difuture.de), LMU Munich, Munich, Germany **4** Munich Center for Machine Learning, Munich, Germany

DOI: 10.21105/joss.04545

Software

- Review
- Repository
- Archive

Editor: Charlotte Soneson

Reviewers:

- @patRyserWelch8
- @brunomontezano
- @AnthonyOfSeattle

Submitted: 13 June 2022
Published: 21 February 2023

Summary

Our R (R Core Team, 2021) package `dsBinVal` implements the methodology explained by Schalk et al. (2022). It extends the ROC-GLM (Pepe, 2000) to distributed data by using techniques of differential privacy (Dwork et al., 2006) and the idea of sharing highly aggregated values only. The package also exports functionality to calculate distributed calibration curves and assess the calibration. Using the package allows us to evaluate a prognostic model based on a binary outcome using the DataSHIELD (Gaye et al., 2014) framework. Therefore, the main functionality makes it able to 1) compute the receiver operating characteristic (ROC) curve using the ROC-GLM from which 2) the area under the curve (AUC) and confidence intervals (CI) are derived to conduct hypothesis testing according to DeLong et al. (1988). Furthermore, 3) the calibration can be assessed distributively via calibration curves and the Brier score. Visualizing the approximated ROC curve, the AUC with confidence intervals, and the calibration curves using `ggplot2` is also supported. Examples can be found in the README file of the repository.

Statement of need

Privacy protection of patient data plays a major role for a variety of tasks in medical research. Uncontrolled release of health information may cause personal disadvantages for individuals, and the individual patient needs to be protected against personal details becoming visible to people not authorized to know them.

In statistics or machine learning, one of these tasks is to gain insights by building statistical or prognostic models. Prognoses on the development of severe health conditions and covariates encoding critical health information, such as genetic susceptibility, need to be handled with care. Furthermore, using confidential data comes with administrative burdens and mostly requires a consent around data usage. Additionally, the data can be distributed over multiple sites (e.g. hospitals) which makes their access even more challenging. Modern approaches in distributed analysis allow work on distributed confidential data by providing frameworks that allow retrieval of information without sharing of sensitive information. Since no sensitive information is shared through the use of privacy-preserving and distributed algorithms, their use helps to meet administrative, ethical, and legal requirements in medical research as users do not have access to personal data.

One of these frameworks for privacy protected analysis is DataSHIELD (Gaye et al., 2014). It allows the analysis of data in a non-disclosive setting. The framework already provides techniques for descriptive statistics, basic summary statistics, and basic statistical modeling.

Within a multiple sclerosis use case to enhance patient medication in the DIFUTURE consortium of the German Medical Informatics Initiative (Prasser et al., 2018), a prognostic model was developed on individual patient data. One goal of the multiple sclerosis use case is to validate that prognostic model using ROC and calibration analysis on patient data distributed across five hospitals using DataSHIELD.

In this package we close the gap between distributed model building and the validation of binary outcomes also on the distributed data. Therefore, our package seamlessly integrates into the DataSHIELD framework, which does not yet provide distributed ROC analysis and calibration assessment.

Functionality

The integration of the `dsBinVal` package into the DataSHIELD framework extends its functionality and allows users to assess the discrimination and calibration of a binary classification model without harming the privacy of individuals. Based on privacy-preserving distributed algorithms (Schalk et al., 2022), the assessment of the discrimination is done by the `dsROCGLM()` function that calculates a ROC curve based on the ROC-GLM as well as an AUC with CI. The calibration is estimated distributively using the functions `dsBrierScore()` and `dsCalibrationCurve()`. Additional helper functions, `dsConfusion()` or `dsL2Sens()`, can be used to calculate several measures, e.g. sensitivity, specificity, accuracy, or the F1 score, from the confusion matrix or the L2-sensitivity. Note that measures from the confusion matrix may be disclosive for specific thresholds and are therefore checked and protected by DataSHIELDs privacy mechanisms. During the call to `dsROCGLM()`, parts of the data set are communicated twice, first, to calculate the ROC-GLM based on prediction scores, and second, to calculate the CI of the AUC. In both steps, the information is protected by differential privacy to prevent individuals from re-identification. The amount of noise generated for differential privacy is carefully chosen based on a simulation study that takes the variation of the predicted values into account. We refer to the README file of the repository for a demonstration and usage of the functionality.

Technical details: To ensure the functioning of our package on DataSHIELD, it is constantly unit tested on an active DataSHIELD test instance. The reference, username, and password are available at the OPAL documentation in the "Types" section. Parts of the tests also cover checks against privacy breaches by attempting to call functions with data sets that do not pass the safety mechanisms of DataSHIELD. Hence, individual functions attempt to prevent accidental disclosures when data is not sufficient to ensure privacy.

State of the field: To the best of our knowledge, there is no distributed ROC-GLM implementation available in `R`. Current state-of-the-art techniques require sharing of sensitive information from the sites and using existing implementation such as `pROC` (Robin et al., 2011) for the ROC curve or standard software for the GLM to calculate the ROC-GLM (as stated by Pepe (2000)).

Acknowledgements

This work was supported by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01IS18036A and Federal Ministry for Research and Technology (BMFT) under Grant No. 01ZZ1804C (DIFUTURE, MII). The authors of this work take full responsibilities for its content.

References

DeLong, E. R., DeLong, D. M., & Clarke-Pearson, D. L. (1988). Comparing the areas under two or more correlated receiver operating characteristic curves: A nonparametric approach.

Biometrics, 837–845. https://doi.org/10.2307/2531595

Dwork, C., McSherry, F., Nissim, K., & Smith, A. (2006). Calibrating noise to sensitivity in private data analysis. *Theory of Cryptography Conference*, 265–284. https://doi.org/10.1007/11681878_14

Gaye, A., Marcon, Y., Isaeva, J., LaFlamme, P., Turner, A., Jones, E. M., Minion, J., Boyd, A. W., Newby, C. J., Nuotio, M.-L., & others. (2014). DataSHIELD: Taking the analysis to the data, not the data to the analysis. *International Journal of Epidemiology*, *43*(6), 1929–1944. https://doi.org/10.1093/ije/dyu188

Pepe, M. S. (2000). An interpretation for the ROC curve and inference using GLM procedures. *Biometrics*, *56*(2), 352–359. https://doi.org/10.1111/j.0006-341x.2000.00352.x

Prasser, F., Kohlbacher, O., Mansmann, U., Bauer, B., & Kuhn, K. A. (2018). Data integration for future medicine (DIFUTURE). *Methods of Information in Medicine*, *57*(S01), e57–e65. https://doi.org/10.3414/ME17-02-0022

R Core Team. (2021). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Robin, X., Turck, N., Hainard, A., Tiberti, N., Lisacek, F., Sanchez, J.-C., & Müller, M. (2011). pROC: An open-source package for R and S+ to analyze and compare ROC curves. *BMC Bioinformatics*, *12*, 77. https://doi.org/10.1186/1471-2105-12-77

Schalk, D., Hoffmann, V. S., Bischl, B., & Mansmann, U. (2022). *Distributed non-disclosive validation of predictive models by a modified ROC-GLM*. arXiv. https://doi.org/10.48550/ARXIV.2203.10828

Part III - Conclusion and Outlook

CHAPTER 12

Conclusion

As described in Chapter I, based on the base learners, CWB is a method that produces flexible and interpretable ML algorithms. In this thesis, three adaptions were presented to adapt CWB for modern needs.

Increasing the efficiency in terms of runtime and memory consumption was shown in Chapter 7, leading to two major scientific contributions. First, introducing ACWB as a CWB variant with Nesterov momentum as an optimizer can speed up the fitting process without losing performance or estimating capabilities. Chapter 7 showed that ACWB could also worsen the performance if not stopped early. Thus, HCWB was proposed by applying early stopping to ACWB and continuing with CWB to fine-tune the model. The reasoning here is to combine the fast convergence of ACWB at the beginning of the fitting process and finishing with CWB to prevent the algorithm from overfitting. The second contribution was to outline how binning can be used in combination with CWB to reduce the memory consumption of the algorithm. Binning discretizes numerical features and allows the use of optimized algorithms to work on that reduced representation. Simulation studies and a benchmark were conducted to underpin the effectiveness of the proposed contributions. The results showed that ACWB/HCWB and binning could tremendously improve the computational efficiency of CWB without losing performance and deteriorating the capability to estimate partial feature effects and provide an (unbiased) feature selection. The software package `compboost` was introduced to increase the efficiency on the software side. To that end, `compboost` uses `C++` as the core language and focuses only on the essential parts required to fit CWB.

`Autocompboost`, presented in Chapter 8, is a novel interpretable AutoML framework based on CWB. `Autocompboost` helps non-expert users to fit CWB and apply good practices of ML. The contribution extends beyond the implementation by providing information about required complexity, decomposing predictions to understand the decision-making, and (if dropping the third stage) explaining univariate partial feature effects as well as pairwise interactions. The information about the required complexity is based on a three-stage construction, with CWB as the fitting engine that allows assessing the performance gain whenever adding a new component.

Chapter 9 explains an algorithm that enables fitting CWB to distributed data. The contribution is a distributed, privacy-preserving, and lossless CWB algorithm with site-specific corrections. Fitting the distributed CWB algorithm relies on aggregated data communicated from the sites to ensure privacy. The proposed algorithm also accounts for site-specific effects, defined as deviations from the main effect that is common for all sites. This corresponds to fitting GAMMs with the sites as a statistical unit with repeated measurements. Hence, being able to fit CWB distributively with site-specific effects also allows fitting GAMMs in a distributed, privacy-preserving, and lossless fashion. The distributed algorithm was compared to the state-of-the-art method on pooled data and showed similar results regarding effect estimation while being privacy preserving and lossless.

Additionally, part of the thesis was to develop methods for distributed computing in a medical context. The background is a study that aims to develop a treatment decision score for multiple sclerosis patients. Evaluating the score with the AUC is one goal of the study. With this objective, Chapter 10 contributes methods to conduct a distributed ROC analysis in a privacy-preserving fashion. Approximating the ROC curve with the ROC-GLM allows the application of a distributed GLM estimation. During the ROC estimation, DP and data aggregations are used as privacy mechanisms. Further, simulations were conducted to assess the approximation error of the distributed approach depending on the privacy parameters ε and δ as well as on the ℓ_2-sensitivity of $\hat{f}$.

CHAPTER 13

Outlook

The presented extensions of CWB are realizations of many potential improvements of CWB. Future valuable efforts may focus on efficiently incorporating pairwise interactions. Including all pairwise interactions often makes fitting CWB infeasible, since $0.5p(p-1)$ base learners must be added. Preselecting feature combinations as potential candidates requires using domain knowledge or an external filter to extract that information. Hence, automatically and efficiently selecting pairwise feature interactions using the capabilities of CWB in the form of base learners could highly increase the performance while maintaining the merits of CWB. Of course, this method could be combined with binning, which is not yet supported for base learners that model multiple features.

In the case of automation, `Autocompboost` is a prototype that requires further investigation to combine different model classes and adequately maintain interpretability. Hence, entirely relying on the partial feature effects estimated in stages one and two requires ensuring that the third stage does not revert the previously estimated effects. Future work will focus on that third stage and how to comply with the interpretability properties of the first two stages. Furthermore, `Autocompboost` would also highly benefit from automatically and efficiently finding pairwise interactions that do not require external intervention.

The proposed distributed CWB algorithm is restricted to only function with horizontally distributed data. Extending the algorithm to additionally handle vertically distributed data[1] highly increases the flexibility in processing heterogeneously distributed data. An algorithm able to estimate and differentiate between main and site-specific effects on both horizontally and vertically distributed data would solve problems such as processing data with inconsistencies of collected features at the sites. Consider an example where a specific sensor is unavailable at one or more hospitals. This requires eliminating the feature from all data sets at these sites. Instead of removing the whole feature from all collaborating sites, a distributed version of CWB could account for missing features at specific sites and allows estimating main and site-specific effects. The hospitals with the missing feature could benefit from using the main effect while the other sites could still use their site-specific adaption for more accurate predictions.

Besides the data modeling, the role of distributed model evaluation will become more important as computing on edge devices becomes more prominent for users of statistics and ML. Therefore, developing a theory for distributed resampling based on existing resampling strategies and backing them up with security mechanisms is a potentially impactful future project.

Finally, all presented and future adaptions could be combined into one AutoML framework for distributed data, with `Autocompboost` as a starting point. The distributed CWB algorithm would work

[1]For vertically distributed data, the features are split and distributed over the sites instead of the observations.

as a fitting engine and facilitates high-quality insights into the data-generating process while preserving data privacy. Furthermore, distributed evaluation allows not just estimating the performance of the final model, but also the generation of further insights about complexity based on the supported performance measures.

References

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318.

Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al. (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58:82–115.

Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and Van der Vorst, H. (1994). *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM.

Bates, D., Maechler, M., and Jagan, M. (2022). *Matrix: Sparse and Dense Matrix Classes and Methods*. R package version 1.5-1.

Bekkerman, R., Bilenko, M., and Langford, J. (2011). *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press.

Biau, G., Cadre, B., and Rouvìère, L. (2019). Accelerated gradient boosting. *Machine Learning*, 108(6):971–992.

Bischl, B., Binder, M., Lang, M., Pielok, T., Richter, J., Coors, S., Thomas, J., Ullmann, T., Becker, M., Boulesteix, A.-L., et al. (2021). Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. *arXiv preprint arXiv:2107.05847*.

Bischl, B., Mersmann, O., Trautmann, H., and Weihs, C. (2012). Resampling methods for meta-model validation with recommendations for evolutionary computation. *Evolutionary computation*, 20:249–75.

Bost, R., Popa, R. A., Tu, S., and Goldwasser, S. (2014). Machine learning classification over encrypted data. Cryptology ePrint Archive, Paper 2014/331. `https://eprint.iacr.org/2014/331`.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brier, G. W. et al. (1950). Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3.

Brockhaus, S., Rügamer, D., and Greven, S. (2020). Boosting functional regression models with fdboost. *Journal of Statistical Software*, 94(10):1–50.

Bühlmann, P., Hothorn, T., et al. (2007). Boosting algorithms: Regularization, prediction and model fitting. *Statistical science*, 22(4):477–505.

Bühlmann, P. and Yu, B. (2003). Boosting with the L2 loss: regression and classification. *Journal of the American Statistical Association*, 98(462):324–339.

Buluc, A. and Gilbert, J. R. (2008). Challenges and advances in parallel sparse matrix-matrix multiplication. In *2008 37th International Conference on Parallel Processing*, pages 503–510.

Casalicchio, G. (2019). *On benchmark experiments and visualization methods for the evaluation and interpretation of machine learning models*. PhD dissertation, LMU Munich.

Chen, Y.-R., Rezapour, A., and Tzeng, W.-G. (2018). Privacy-preserving ridge regression on distributed data. *Information Sciences*, 451:34–49.

Choi, J., Walker, D. W., and Dongarra, J. J. (1994). Pumma: Parallel universal matrix multiplication algorithms on distributed memory concurrent computers. *Concurrency: Practice and Experience*, 6(7):543–570.

Coors, S., Schalk, D., Bischl, B., and Rügamer, D. (2021). Automatic componentwise boosting: An interpretable automl system. *ECML-PKDD Workshop on Automating Data Science*.

Cunha, M., Mendes, R., and Vilela, J. P. (2021). A survey of privacy-preserving mechanisms for heterogeneous data types. *Computer Science Review*, 41:100403.

Dagum, L. and Menon, R. (1998). Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55.

Davis, T. A. (2006). *Direct methods for sparse linear systems*. SIAM.

Drozdal, J., Weisz, J., Wang, D., Dass, G., Yao, B., Zhao, C., Muller, M., Ju, L., and Su, H. (2020). Trust in automl: Exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, IUI '20, page 297–307, New York, NY, USA. Association for Computing Machinery.

Duff, I. S., Grimes, R. G., and Lewis, J. G. (1989). Sparse matrix test problems. *ACM Transactions on Mathematical Software (TOMS)*, 15(1):1–14.

Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., and Naor, M. (2006a). Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 486–503. Springer.

Dwork, C., McSherry, F., Nissim, K., and Smith, A. (2006b). Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer.

Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407.

Eilers, P. H. and Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. *Statistical science*, pages 89–102.

Fang, H. and Qian, Q. (2021). Privacy preserving machine learning with homomorphic encryption and federated learning. *Future Internet*, 13(4).

Fawcett, T. (2006). An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874.

Feurer, M. and Hutter, F. (2019). Hyperparameter optimization. In *Automated machine learning*, pages 3–33. Springer, Cham.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28.

Flach, P. (2012). *Machine learning: the art and science of algorithms that make sense of data*. Cambridge university press.

Freitas, A. A. (2019). Automated machine learning for studying the trade-off between predictive accuracy and interpretability. In Holzinger, A., Kieseberg, P., Tjoa, A. M., and Weippl, E., editors, *Machine Learning and Knowledge Extraction*, pages 48–66, Cham. Springer International Publishing.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232.

Gambs, S., Kégl, B., and Aïmeur, E. (2007). Privacy-preserving boosting. *Data Mining and Knowledge Discovery*, 14(1):131–170.

Gaye, A., Marcon, Y., Isaeva, J., LaFlamme, P., Turner, A., Jones, E. M., Minion, J., Boyd, A. W., Newby, C. J., Nuotio, M.-L., et al. (2014). Datashield: taking the analysis to the data, not the data to the analysis. *International journal of epidemiology*, 43(6):1929–1944.

Gong, M., Xie, Y., Pan, K., Feng, K., and Qin, A. (2020). A survey on differentially private machine learning [review article]. *IEEE Computational Intelligence Magazine*, 15(2):49–64.

Gordon, D. F. and Desjardins, M. (1995). Evaluation and selection of biases in machine learning. *Machine learning*, 20(1):5–22.

Hastie, T. J. (2017). Generalized additive models. In *Statistical models in S*, pages 249–307. Routledge.

Hofner, B., Hothorn, T., Kneib, T., and Schmid, M. (2011). A framework for unbiased model selection based on boosting. *Journal of Computational and Graphical Statistics*, 20(4):956–971.

Hofner, B., Mayr, A., and Schmid, M. (2016). gamboostLSS: An R package for model building and variable selection in the GAMLSS framework. *Journal of Statistical Software*, 74(1).

Hothorn, T., Bühlmann, P., Kneib, T., Schmid, M., and Hofner, B. (2010). Model-based boosting 2.0. *The Journal of Machine Learning Research*, 11:2109–2113.

Hothorn, T., Bühlmann, P., Kneib, T., Schmid, M., and Hofner, B. (2020). *mboost: Model-based boosting*. R package version 2.9-7.

Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). *Automated machine learning: methods, systems, challenges*. Springer Nature.

Jayaraman, B. and Evans, D. (2019). Evaluating differentially private machine learning in practice. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1895–1912.

John, G. H. (1995). Robust decision trees: Removing outliers from databases. In *KDD*, volume 95, pages 174–179.

Karr, A. F., Lin, X., Sanil, A. P., and Reiter, J. P. (2005). Secure regression on distributed databases. *Journal of Computational and Graphical Statistics*, 14(2):263–279.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 18(25):1–5.

Lang, S., Umlauf, N., Wechselberger, P., Harttgen, K., and Kneib, T. (2014). Multilevel structured additive regression. *Statistics and Computing*, 24(2):223–238.

Lazarevic, A. and Obradovic, Z. (2001). The distributed boosting algorithm. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 311–316.

Li, J., Kuang, X., Lin, S., Ma, X., and Tang, Y. (2020). Privacy preservation for machine learning training and classification based on homomorphic encryption schemes. *Information Sciences*, 526:166–179.

Li, Y., Jiang, X., Wang, S., Xiong, H., and Ohno-Machado, L. (2016). Vertical grid logistic regression (vertigo). *Journal of the American Medical Informatics Association*, 23(3):570–579.

Li, Z. and Wood, S. N. (2020). Faster model matrix crossproducts for large generalized linear models with discretized covariates. *Statistics and Computing*, 30(1):19–25.

Liew, B. X., Rügamer, D., Abichandani, D., and De Nunzio, A. M. (2020a). Classifying individuals with and without patellofemoral pain syndrome using ground force profiles – Development of a method using functional data boosting. *Gait & Posture*, 80:90–95.

Liew, B. X., Rügamer, D., Stocker, A., and De Nunzio, A. M. (2020b). Classifying neck pain status using scalar and functional biomechanical variables – Development of a method using functional data boosting. *Gait & posture*, 76:146–150.

Lu, H., Karimireddy, S. P., Ponomareva, N., and Mirrokni, V. (2020). Accelerating gradient boosting machines. In *International Conference on Artificial Intelligence and Statistics*, pages 516–526. PMLR.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Luo, C., Islam, M., Sheils, N. E., Buresh, J., Reps, J., Schuemie, M. J., Ryan, P. B., Edmondson, M., Duan, R., Tong, J., et al. (2022). Dlmm as a lossless one-shot algorithm for collaborative multi-site distributed linear mixed models. *Nature Communications*, 13(1):1–10.

Machanavajjhala, A., Kifer, D., Gehrke, J., and Venkitasubramaniam, M. (2007). l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and Arcas, B. A. y. (2017). Communication-Efficient Learning of Deep Networks from Decentralized Data. In Singh, A. and Zhu, J., editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR.

Mohassel, P. and Zhang, Y. (2017). Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE.

Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

Nesterov, Y. (1983). A method for solving the convex programming problem with convergence rate $O(1/k^2)$.

Pepe, M. S. (2000). An interpretation for the roc curve and inference using glm procedures. *Biometrics*, 56(2):352–359.

Pepe, M. S. (2003). The statistical evaluation of medical tests for classification and prediction. *Journal of the American Statistical Association*.

Pfisterer, F. (2022). *Democratizing Machine Learning – Contributions in AutoML and Fairness*. PhD thesis, LMU Munich.

Prasser, F., Kohlbacher, O., Mansmann, U., Bauer, B., and Kuhn, K. A. (2018). Data integration for future medicine (difuture). *Methods Inf Med*, 57(S01):e57–e65.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215.

Rügamer, D., Brockhaus, S., Gentsch, K., Scherer, K., and Greven, S. (2018). Boosting factor-specific functional historical models for the detection of synchronization in bioelectrical signals. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 67(3):621–642.

Saintigny, P., Zhang, L., Fan, Y.-H., El-Naggar, A. K., Papadimitrakopoulou, V. A., Feng, L., Lee, J. J., Kim, E. S., Hong, W. K., and Mao, L. (2011). Gene expression profiling predicts the development of oral cancer. *Cancer Prevention Research*, 4(2):218–229.

Sanderson, C. and Curtin, R. (2016). Armadillo: a template-based c++ library for linear algebra. *Journal of Open Source Software*, 1(2):26.

Sanderson, C. and Curtin, R. (2018). A user-friendly hybrid sparse matrix class in c++. In *International Congress on Mathematical Software*, pages 422–430. Springer.

Schalk, D., Bischl, B., and Rügamer, D. (2022a). Accelerated componentwise gradient boosting using efficient data representation and momentum-based optimization. *Journal of Computational and Graphical Statistics*.

Schalk, D., Bischl, B., and Rügamer, D. (2023a). Privacy-preserving and lossless distributed estimation of high-dimensional generalized additive mixed models. *arXiv preprint arXiv:2210.07723*.

Schalk, D., Hoffmann, V. S., Bischl, B., and Mansmann, U. (2022b). Distributed non-disclosive validation of predictive models by a modified roc-glm. *arXiv preprint arXiv:2203.10828*.

Schalk, D., Hoffmann, V. S., Bischl, B., and Mansmann, U. (2023b). dsBinVal: Conducting distributed roc analysis using datashield. *Journal of Open Source Software*, 8(82):4545.

Schalk, D., Thomas, J., and Bischl, B. (2018). compboost: Modular framework for component-wise boosting. *Journal of Open Source Software*, 3(30):967.

Schmid, M. and Hothorn, T. (2008). Boosting additive models using component-wise p-splines. *Computational Statistics & Data Analysis*, 53(2):298–311.

Shahnaz, R., Usman, A., and Chughtai, I. R. (2005). Review of storage techniques for sparse matrices. In *2005 Pakistan Section Multitopic Conference*, pages 1–7.

Stehman, S. V. (1997). Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89.

Sun, X., Zhang, P., Liu, J. K., Yu, J., and Xie, W. (2020). Private machine learning classification based on fully homomorphic encryption. *IEEE Transactions on Emerging Topics in Computing*, 8(2):352–364.

Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International journal of uncertainty, fuzziness and knowledge-based systems*, 10(05):557–570.

Thomas, J., Coors, S., and Bischl, B. (2018). Automatic gradient boosting. *ICML AutoML Workshop*.

Thomas, J., Hepp, T., Mayr, A., and Bischl, B. (2017). Probing for sparse and fast variable selection with model-based boosting. *Computational and mathematical methods in medicine*, 2017.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855.

Tutz, G. and Gertheiss, J. (2016). Regularized regression for categorical data. *Statistical Modelling*, 16(3):161–200.

Van Buuren, S. (2018). *Flexible imputation of missing data*. CRC press.

Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., and Rellermeyer, J. S. (2020). A survey on distributed machine learning. *Acm computing surveys (csur)*, 53(2):1–33.

Wang, Q. and Kurz, D. (2022). Reconstructing training data from diverse ml models by ensemble inversion. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2909–2917.

Wood, S. N. (2017). *Generalized additive models: an introduction with R*. Chapman and Hall/CRC.

Wood, S. N., Li, Z., Shaddick, G., and Augustin, N. H. (2017). Generalized additive models for gigadata: Modeling the u.k. black smoke network daily data. *Journal of the American Statistical Association*, 112(519):1199–1210.

Xanthopoulos, I., Tsamardinos, I., Christophides, V., Simon, E., and Salinger, A. (2020). Putting the human back in the automl loop. In *EDBT/ICDT Workshops*.

Yan, Z., Zachrison, K. S., Schwamm, L. H., Estrada, J. J., and Duan, R. (2022). Fed-glmm: A privacy-preserving and computation-efficient federated algorithm for generalized linear mixed models to analyze correlated electronic health records data. *medRxiv*.

Zhu, R., Jiang, C., Wang, X., Wang, S., Zheng, H., and Tang, H. (2020). Privacy-preserving construction of generalized linear mixed model for biomedical computation. *Bioinformatics*, 36(Supplement_1):i128–i135.

Eidesstattliche Versicherung

(Siehe Promotionsordnung vom 12. Juli 2011, §8 Abs. 2 Pkt. 5)

Hiermit erkläre ich an Eides statt, dass die Dissertation von mir selbstständig,
ohne unerlaubte Beihilfe angefertigt ist.

München, den 07.12.2022

Daniel Schalk