Gaussian Process Boosting

Fabio Sigrist[*]
Lucerne University of Applied Sciences and Arts

December 25, 2020

Abstract

We introduce a novel way to combine boosting with Gaussian process and mixed effects models. This allows for relaxing, first, the linearity assumption for the mean function in Gaussian process and mixed effects models in a flexible non-parametric way and, second, the independence assumption made in most boosting algorithms. The former is advantageous for predictive accuracy and for avoiding model misspecifications. The latter is important for more efficient learning of the mean function and for obtaining probabilistic predictions. In addition, we present an extension that scales to large data using a Vecchia approximation for the Gaussian process model relying on novel results for covariance parameter inference. We obtain increased predictive accuracy compared to existing approaches on several simulated and real-world data sets.

Keywords: Non-linear mixed effects models, random effects, longitudinal data, panel data, spatial data, temporal data, boosting with high-cardinality categorical variables, machine learning

1 Introduction

Both boosting and Gaussian process and mixed effects models are frequently used in applied machine learning and statistics. To date, these methods have been considered as separate approaches with each having its advantages. In this article, we propose a novel way to combine boosting with Gaussian process and mixed effects models.

Boosting [Freund and Schapire, 1996, Breiman, 1998, Friedman et al., 2000, Mason et al., 2000, Friedman, 2001, Bühlmann and Hothorn, 2007] is a machine learning technique that achieves superior predictive performance for a large variety of data sets [Chen and Guestrin, 2016, Nielsen, 2016]. This is reflected in statements such as "[i]n general 'boosted decision trees' is regarded as the most effective off-the-shelf nonlinear learning method for a wide range of application problems" [Johnson and Zhang, 2013]. Apart from this, the wide adoption of tree-boosting in applied machine learning and data science is due to several advantages: boosting with trees as base learners can automatically account for complex non-linearities, discontinuities, and high-order interactions, it is robust to outliers in and multicollinearity among predictor variables, it is scale-invariant to monotone transformations of the predictor variables, it can handle missing values in predictor variables automatically by loosing almost no information [Elith et al., 2008], and boosting can perform variable selection. Gaussian processes [Williams and Rasmussen, 2006] are flexible non-parametric function models that achieve state-of-the-art predictive accuracy and allow for making probabilistic predictions [Gneiting et al., 2007]. They are used in areas such non-parametric regression, modeling of

[*]Email: fabio.sigrist@hslu.ch. Address: Lucerne University of Applied Sciences and Arts, Suurstoffi 1, 6343 Rotkreuz, Switzerland.

time series [Shumway and Stoffer, 2017], spatial [Banerjee et al., 2014], and spatio-temporal [Cressie and Wikle, 2015] data, emulation of large computer experiments [Kennedy and O'Hagan, 2001], optimization of expensive black-box functions [Jones et al., 1998], and parameter tuning in machine learning models [Snoek et al., 2012]. Further, mixed effects models with grouped, or clustered, random effects are widely used is various scientific disciplines for modeling panel and longitudinal data or, in general, data with a hierarchically nested or crossed grouping structure [Pinheiro and Bates, 2006].

In Gaussian process and mixed effects models, the first moment is often assumed to be either zero or a linear function of covariates. Residual structured variation is then modeled using a zero-mean Gaussian process and/or grouped random effects model. However, both the zero-mean and the linearity assumption are often unrealistic, and higher predictive accuracy can be obtained by relaxing these assumptions; see, e.g., our experiments in Sections 4 and 5. Furthermore, if the mean function of a Gaussian process model is misspecified, spurious second-order non-stationarity can occur as the covariance function of such a misspecified model equals the true covariance function plus the squared bias of the mean function [Fuglstad et al., 2015, Schmidt and Guttorp, 2020]. It is thus important to first correctly model the mean function before trying to account for potential residual second-order non-stationarity.

In many state-of-the-art supervised machine learning algorithms, in particular in boosting, it is assumed that a flexible and potentially complex function relates a set of predictor variables to a response variable. Conditional on the predictor variables, the response variable is assumed to be independent across observations. This means that potential residual correlation, i.e. correlation that is not accounted for by the regression function, is ignored. As we show in our experiments in Sections 4 and 5, modeling such correlation allows not only for better learning of the regression function, but it is also important for prediction, in particular for probabilistic predictions and for predicting averages, or sums, over space (block kriging), time, and groups or clusters. Examples of this include the prediction of global average temperatures, the total rainfall in a catchment area, the total value of a portfolio of real estate objects, or the average price of products offered by multiple sellers.

In summary, the goal of this article is to relax both the linearity assumption in Gaussian process and mixed effects models and the independence assumption in boosting by combining these two approaches. This is done by considering a mixed effects model where the mean function is assumed to be a non-parametric and non-linear regression function, and the random effects can consist of various combinations of Gaussian processes for, e.g., spatial and/or temporal data, as well as hierarchically clustered, nested, crossed, and random coefficient effects. Specifically, we propose to model the mean function by an ensemble of base learners, such as regression trees [Breiman et al., 1984], learned in a stage-wise manner using boosting, and the parameters of the covariance structure of the random effects are jointly estimated with the mean function; see Section 2 for more details.

1.1 Relation to existing work

We adopt the terminology used in the mixed effects models literature [Laird et al., 1982, Pinheiro and Bates, 2006] in this article; see Section 2.1 for more information. The majority of the existing Gaussian process and mixed effects models assume that the mean function is a linear regression function. Little research has been done on combining modern supervised machine learning techniques, such as (tree-)boosting or random forests, with mixed effects models and, in particular, Gaussian processes. In the following, we give a brief review of existing literature for mixed effects models where a non-linear mean function is estimated in a flexible, non-parametric way focusing on approaches that do not make prior assumptions on the structure of the functional form of the mean function.

To relax the linearity assumption in mixed effects models, Tutz and Reithinger [2007] and Groll and Tutz [2012] propose to use generalized additive models [Hastie and Tibshirani, 1986, Wood, 2017]. However, the structure of the mean function has to be determined a priori by specifying, for instance, main and second-order interaction effects. In general, this can thus result in model misspecification. For the special case of grouped random effects models for clustered or longitudinal data, several non-parametric, machine learning-based approaches have been proposed. This includes Hajjem et al. [2011], Sela and Simonoff [2012], and Fu and Simonoff [2015] which use regression trees for the mean function, and Hajjem et al. [2014] which relies on random forest to model the mean function. Both the MERT and MERF approaches of Hajjem et al. [2011] and Hajjem et al. [2014] and the RE-EM tree algorithms of Sela and Simonoff [2012] and Fu and Simonoff [2015] repeatedly (i) learn the mean function using a machine learning technique, (ii) estimate covariance parameters, and (iii) calculate predictions for random effects. This can make these approaches computationally demanding, in particular when the sample size is large and the mean function is modeled using a complex model since both covariance parameters and the fixed effects functions are repeatedly learned in every iteration. What is more, despite the suggestions of their names, these algorithms [Sela and Simonoff, 2012, Fu and Simonoff, 2015, Hajjem et al., 2011, 2014] are heuristically motivated and do not correspond to correctly specified EM algorithms for mixed effects models [Laird et al., 1982] as they do not contain a proper E-step. Since it is nowhere clearly stated in the previous literature which optimization problems are solved by these algorithms, we elaborate on this in Section A in the appendix. In particular, it is unclear whether and to which quantities the MERT and MERF algorithms [Hajjem et al., 2011, 2014] converge. In our experiments in Sections 4 and 5, we do not obtain convergence for the MERF algorithm. Recently, Pande et al. [2017] propose a tree-boosting approach for a special type of mixed effects models for longitudinal data with single-level grouped random effects and where predictor variables are assumed to be constant within subjects. Focusing on modeling complex interactions between time and predictor variables, their approach differs from ours in several directions. For instance, they do not cover Gaussian processes or other forms of random effects models with complex clustering such as crossed or nested random effects. Furthermore, they (re-)estimate the covariance parameters in every boosting iteration using the `nlme` R package, and their boostmtree algorithm uses a certain form of "in-sample cross-validation" to avoid overfitting.

A straightforward alternative to combining Gaussian process and mixed effects models with a non-parametric mean function consists of using any statistical or machine learning approach, such as boosting or random forest, and simply including the locations which define the Gaussian process[1] as continuous variables or the grouping variables for grouped random effects as categorical variables in the set of predictor variables for the mean function. For linear models, this is equivalent to using fixed effects instead of random effects. However, this approach has several drawbacks. First, when modeling spatial data, it is often required that the spatial effect is continuous over space, but tree-boosting and random forest produce discontinuous functions. A way to avoid this problem in boosting is to use base learners that are continuous in the locations. This is the approach proposed in Hothorn et al. [2010] where splines are used to model spatial effects and ridge regression is used to model grouped, or clustered, effects. As we argue in the following, this approach still has various drawbacks. First, splines have the disadvantage that they suffer from the so-called "curse of dimensionality" when the dimension of the "locations" is large and the locations are thus sparse in space; see the beginning of Section 1 for examples where this occurs. Moreover, all boosting approaches, irrespective of the base learners used, which model spatial or grouped effects using deterministic base learners have the drawback that they assume a deterministic relationship and that the residual error term is the only source of variation. On the other hand, Gaussian process and mixed effects models assume probabilistic models for the spatial and grouped effects and can thus provide probabilistic predictions accounting for uncertainty in prediction. This is particularly important for probabilistic predictions of, e.g., areal sums or averages as correlation between the different random effects needs to be taken into account. The latter cannot be done by a boosting approach assuming independence conditional on deterministic base learners. Besides, in the case of linear models, estimates are usually less efficient when using fixed effects instead of random effects.

[1]We follow the spatial statistics terminology and use the term "locations", but these can equally well consists of, e.g., time points for data with time stamps or generally other variables that define a Gaussian process. In machine learning, the locations are usually called "features"; see Section 2.2.2 for more details.

It is thus likely that the mean function is also less efficiently estimated in such a fixed-effects boosting approach, or any other purely fixed-effects machine learning technique. Our simulated experiments in Section 4 support this hypothesis. This is related to the fact that classical tree-boosting algorithms have difficulties with high-cardinality categorical variables. Our proposed approach can thus also be seen as a novel solution for tree-boosting with high-cardinality categorical variables.

2 A non-linear and non-parametric mixed effects model

2.1 Model assumptions

Following the terminology used in the mixed effects models literature [Laird et al., 1982, Pinheiro and Bates, 2006], we assume a mixed effects model of the form

$$y = F(X) + Zb + \epsilon, \quad b \sim \mathcal{N}(0, \Sigma), \quad \epsilon \sim \mathcal{N}(0, \sigma^2 I_n), \tag{1}$$

where $y = (y_1, \dots, y_n)^T \in \mathbb{R}^n$ is the response variable, $F(X) \in \mathbb{R}^n$ are the so-called fixed effects, $b \in \mathbb{R}^m$ are the random effects with covariance matrix $\Sigma \in \mathbb{R}^{m\times m}$, and $\epsilon = (\epsilon_1, \dots, \epsilon_n)^T \in \mathbb{R}^n$ is an independent error term. Specifically, $F(X)$ is the row-wise evaluation of a function $F : \mathbb{R}^p \to \mathbb{R}$, i.e., $F(X) = (F(X_1), \dots, F(X_n))^T$, where $X_i = (X_{i1} \dots, X_{ip})^T \in \mathbb{R}^p$ is the i-th row of X containing predictor variables for observation i, $i = 1, \dots, n$. The matrices $X \in \mathbb{R}^{n\times p}$ and $Z \in \mathbb{R}^{n\times m}$ are the fixed and random effects predictor variable matrices. Further, n denotes the number of data points, m denotes the dimension of the random effects b, and p denotes the number of predictor variables in X.

The random effects vector b is assumed to be either a finite-dimensional version of a Gaussian process and/or to contain grouped random effects.[2] The covariance matrix $\text{Cov}(b) = \Sigma$ is usually assumed to be parametrized by a relatively low number of parameters and it can depend on predictor variables $S \in \mathbb{R}^{n\times d}$. For instance, for spatial and temporal Gaussian processes, these predictor variables are locations and time points, respectively. For notational simplicity, we suppress the dependence of Σ on its parameters and also on S. The matrix Z relating the random effects b to the response variable y is often simply an incidence matrix with entries in $\{0, 1\}$ for, e.g., grouped random effects or Gaussian processes with multiple observations at the same location, but it can also contain covariate data, for instance, in the case of random coefficient models. Note that the predictor variables in Z and S may or may not be a subset of the predictor variables in X. In Section 2.2, we outline several examples and special cases of models for Zb that are included in the specification in (1). Note that, conditional on $F(X)$ and Z, dependence among the response variable y can arise either due to the matrix Z being non-diagonal or due to the covariance matrix Σ of the random effects being non-diagonal.

Further, we assume that $F(\cdot)$ is a function in a function space $\mathcal{H}$ which is the linear span of a set $\mathcal{S}$ of so-called base learners $f_j(\cdot) : \mathbb{R}^p \to \mathbb{R}$. Classes of base learners can consist of, e.g., linear functions [Bühlmann et al., 2006], smoothing splines [Bühlmann and Yu, 2003], wavelets [Saberian et al., 2011], reproducing kernel Hilbert space (RKHS) regression functions [Sigrist, 2019], and regression trees [Breiman et al., 1984], with the latter being the most popular choice, in particular, in applied machine learning due to the advantages listed in Section 1. For defining functional derivatives, we further assume that the space $\mathcal{H}$ is normed. For instance, assuming that the X_i's are identically distributed and that all $F \in \mathcal{H}$ are square integrable with respect to the law of X_1, a norm on $\mathcal{H}$ can defined by the inner product $\langle F, G\rangle = E_{X_1}(F(X_1)G(X_1))$ for $F, G \in \mathcal{H}$. Finally, note that if $F(X) = X^T\beta$, where $\beta \in \mathbb{R}^p$ is a vector of coefficients, the model in (1) is called a linear mixed effects model.

[2]Note that we assume that the random effects follow a normal distribution, but moderate violations of this assumption have been shown to have only a small effect on prediction accuracy in the context of generalized linear mixed models [McCulloch and Neuhaus, 2011].

2.2 Examples and special cases of random effects model

In this section, we outline several examples of random effects models. We distinguish between the following broad classes of random effects models: (i) models where the random effects b are defined by some form of hierarchical grouping denoted as grouped random effects models in this article, (ii) Gaussian process models where the random effects are finite-dimensional versions of Gaussian processes, and (iii) combinations of these two types of random effects.

2.2.1 Grouped random effects model

So-called grouped, or clustered, random effects occur if there is a grouping of the data, and the grouped random effects account for correlation due to this grouping structure. Examples of grouped random effects covered by specifications include single-level random effects, crossed and hierarchically nested random effects, and random coefficient effects. Depending on the application area, such models are also denoted as longitudinal, panel data, or repeated measurement models. Basically, every categorical predictor variable can be interpreted as partitioning the data into different groups.

In a single-level grouped random effects model, for instance, it is assumed that there is a hierarchical grouping of several units each having multiple observations which are dependent within units. Specifically, assume that there is a categorical variable that partitions the data into m groups according to the levels of the variable. Then, it is assumed that there is a random effect $b_j \in \mathbb{R}$, $j = 1, \dots, m$, for every group j, and the random effects b_j are assumed to be independent and identically distributed with $\mathrm{Cov}(b) = \Sigma = \sigma_1^2 I_m$. We thus have

$$\Psi = \sigma_1^2 Z Z^T + \sigma^2 I_n,$$

where the matrix Z is an incidence matrix $Z \in \{0,1\}^{n \times m}$ that relates group-level random effects to observations. Such a single-level model can be easily extended to allow for multiple random effects which can be nested or crossed and also consist of random coefficients, i.e. random slopes. In the latter case, Z is no longer a binary incidence matrix but it contains covariate data.

2.2.2 Gaussian process model

In a Gaussian process model, one assumes that the random effects $b = (b(s_1), \dots, b(s_m))^T$ are a finite-dimensional version of a Gaussian process $b(s)$ with a parametric covariance function

$$Cov(b(s), b(s')) = c(s, s'), \quad s, s' \in \mathbb{R}^d,$$

observed at locations $s_1, \dots, s_m$. Note that we use the terminology "locations" since Gaussian processes are widely used in spatial statistics, but the locations can in general also consist of covariates, or features, that do not necessarily correspond to locations in a physical space. In particular, the locations can also contain the covariates X_i or a subset of them. The matrix Z can be either a binary incidence matrix to model multiple observations at the same locations and/or it can contain covariate data for random coefficient Gaussian processes, also called spatially varying coefficient models [Gelfand et al., 2003].

Often, the covariance function is assumed to be second-order stationary, mean-square continuous, and parametrized of the form

$$c(s, s') = \sigma_1^2 r(\|s - s'\|/\rho),$$

where r is an isotropic autocorrelation function with $r(0) = 0$, $\sigma_1^2 = Var(b(s))$, and ρ is a so-called range parameter that determines how fast the autocorrelation decays with distance. Examples of autocorrelation functions include the exponential function $r(\|s - s'\|/\rho) = \exp(-\|s - s'\|/\rho)$ and the Gaussian function $r(\|s - s'\|/\rho) = \exp\left(-(\|s - s'\|/\rho)^2\right)$. The extension to more general covariance functions and also to multivariate Gaussian processes is straightforward and our methodology presented in Section 3 does not rely on, e.g., stationarity assumptions. For a stationary Gaussian process, we obtain the following covariance matrix

$$\Psi = \sigma_1^2 Z\Sigma Z^T + \sigma^2 I_n,$$

where $\Sigma \in \mathbb{R}^{m\times m}$ has entries

$$(\Sigma)_{jk} = \sigma_1^2 r(d_{jk}/\rho)$$

and $d_{jk} = \|s_j - s_k\|$, $j,k = 1,\ldots,m$.

2.2.3 Joint grouped random effects and Gaussian process models

Grouped random effects and Gaussian processes can also be combined. An example where such models are used is so-called repeated measures data. For instance, one can assume that within groups, there is temporal and/or spatial dependence modeled by a Gaussian process. In a single-level grouped random effect model, this means that every group contains a Gaussian process and the different Gaussian processes are independent among each other. Alternatively, one can assume that there is one single global Gaussian process in combination with grouped random effects, i.e. the same Gaussian process is related to all observations and it accounts, for instance, for spatial or temporal correlation among all observations.

2.3 Likelihood and risk functional

In the following, we derive the likelihood for the model in (1) and the risk functional that we optimize. The density of the response y in (1) is given by

$$p(y|F,\theta) = \int p(y|b,F,\theta)p(b|\theta)db, \tag{2}$$

where

$$p(b|\theta) = \exp\left(-\frac{1}{2}b^T\Sigma^{-1}b\right)|\Sigma|^{-1/2}(2\pi)^{-m/2},$$
$$p(y|b,F,\theta) = \exp\left(-\frac{1}{2\sigma^2}(y-F-Zb)^T(y-F-Zb)\right)\left(2\pi\sigma^2\right)^{-n/2},$$

where we abbreviate $F = F(X)$ and $\theta \in \Theta \subset \mathbb{R}^q$ denotes all variance and covariance parameters, i.e. σ^2 and the parameters of Σ. In this article, we assume that the first element of θ is the error variance, $\theta_1 = \sigma^2$. Further, to distinguish a function from its evaluation, we use the symbols "$F(\cdot)$" to denote a function and "F" for the function evaluated at X in the following.

The marginalization in (2) can be done analytically, and the marginal distribution of y is given by

$$y \sim \mathcal{N}\left(F(X),\Psi\right), \quad \Psi = Z\Sigma Z^T + \sigma^2 I_n.$$

This means that the negative log-likelihood of this model is given by

$$L(y,F,\theta) = \frac{1}{2}(y-F)^T\Psi^{-1}(y-F) + \frac{1}{2}\log\det(\Psi) + \frac{n}{2}\log(2\pi). \tag{3}$$

Factoring out the error variance σ^2 by setting

$$\Sigma^\dagger = \Sigma/\sigma^2 \quad \text{and} \quad \Psi^\dagger = \Psi/\sigma^2$$

gives the equivalent form

$$y \sim \mathcal{N}\left(F(X),\sigma^2\Psi^\dagger\right).$$

As this leads to a closed form expression for estimating σ^2 (see Section 3.2), it can be beneficial to use this reparametrization in which, after factoring out σ^2, all other variance parameters are replaced by the ratio of their original value and the error variance σ^2. In particular, calculations in the `gpboost` library (see Section 3.6), which implements our methodology and which we use in the simulated and real-world experiments in this article, are based on this reparametrization.

Finally, our goal is to minimize the risk functional

$$R(F(\cdot),\theta): \quad (F(\cdot),\theta) \;\mapsto\; L(y,F,\theta)\Big|_{F=F(X)}, \tag{4}$$

where $L(y,F,\theta)$ is defined in (3). Note that $R(F(\cdot),\theta)$ is calculated by evaluating $F(\cdot)$ at X and then calculating $L(y,F=F(X),\theta)$. We recall that $F(\cdot)$ is a function in a function space $\mathcal{H}$ and $\theta \in \Theta \subset \mathbb{R}^q$ is the vector of all variance and covariance parameters. This means that the risk functional $R(F(\cdot),\theta)$ is, in general, infinite dimensional in its first argument and finite dimensional in its second argument.

3 Combining Gaussian process and mixed effects models with boosting

Recall that the goal is to find the following joint minimizer

$$(\hat{F}(\cdot),\hat{\theta}) = \underset{(F(\cdot),\theta)\in(\mathcal{H},\Theta)}{\operatorname{argmin}} R(F(\cdot),\theta), \tag{5}$$

where the risk functional $R(F(\cdot),\theta)$ is defined in (4). We propose to do this minimization using a novel boosting algorithm presented in this section.

3.1 Boosting for θ fixed

In the following, we show how boosting works in our case when the variance and covariance parameters θ are given. For fixed θ, boosting finds a minimizer of the empirical risk functional $R(F(\cdot),\theta)$ in a stagewise way by sequentially adding an update $f_m(\cdot)$ to the current estimate $F_{m-1}(\cdot)$:

$$F_m(\cdot) = F_{m-1}(\cdot) + f_m(\cdot), \quad f_m \in \mathcal{S}, \quad m = 1,\dots,M, \tag{6}$$

where $f_m(\cdot)$ is chosen in a way such that its addition results in the minimization of the risk. This minimization can usually not be done analytically and, consequently, an approximation is used. In most boosting algorithms, such an approximation is obtained using either a penalized functional first-order or a functional second-order Taylor expansion of the risk around the current estimate $F_{m-1}(\cdot)$. This then corresponds to functional gradient descent or functional Newton steps. It is also possible to combine gradient and Newton steps [Friedman, 2001] by first learning part of the parameters of the base learners using a gradient step and the remaining part using a Newton update. See Sigrist [2021] for more information on the distinction between gradient, Newton, and hybrid gradient-Newton boosting. Note that, in contrast to existing boosting algorithms, in our case, all observations are dependent in general, i.e., we have only one independent multivariate sample.

In gradient boosting, $f_m(\cdot)$ is given by the least squares approximation to the vector obtained when evaluating the negative functional gradient, i.e. the negative Gâteaux derivative, of $R(F(\cdot),\theta)$ at $(F_{m-1}(\cdot), I_{X_i}(\cdot))$ and X_i, $i=1,\dots,n$, where $I_{X_i}(\cdot)$ are indicator functions which equal 1 at (X_i) and 0 otherwise. Equivalently, one can show that $f_m(\cdot)$ corresponds to a minimizer of a first-order functional Taylor approximation of $R(F_{m-1}(\cdot)+f(\cdot),\theta)$ around $F_{m-1}(\cdot)$ with an L^2 penalty on $f(\cdot)$ evaluated at (X_i), $i=1,\dots,n$; see, e.g., Sigrist [2021] for more information. It is easily seen that the negative Gâteaux derivative of $R(F(\cdot),\theta)$ evaluated at $(F_{m-1}(\cdot), I_{X_i}(\cdot))$ and (X_i), $i=1,\dots,n$, is given by

$$-\frac{\partial}{\partial F} L(y,F,\theta)\Big|_{F=F_{m-1}} = \Psi^{-1}(y - F_{m-1}).$$

In other words, for θ fixed, gradient boosting finds $f_m(\cdot)$ as the least squares approximation

$$f_m(\cdot) = \underset{f(\cdot)\in\mathcal{S}}{\operatorname{argmin}} \left\|\Psi^{-1}(y - F_{m-1}) - f\right\|^2, \tag{7}$$

where $f = (f(X_1), \dots, f(X_n))^T$.

In Newton boosting, $f_m(\cdot)$ is found as the minimizer of a second-order functional Taylor approximation to $R(F_{m-1}(\cdot) + f(\cdot), \theta)$ around $F_{m-1}(\cdot)$. In our case, this gives

$$f_m(\cdot) = \underset{f(\cdot)\in\mathcal{S}}{\operatorname{argmin}} \left(y - F_{m-1} - f\right)^T {\Psi^{\dagger}}^{-1} \left(y - F_{m-1} - f\right). \tag{8}$$

As mentioned, there also exists a hybrid gradient-Newton boosting version which learns part of the parameters of the base learner $f_m(\cdot)$ using a gradient step and the remaining part using a Newton step. In the following paragraph, we assume that the base learners can be written in the form

$$f(\cdot) = h(\cdot;\alpha)^T\gamma, \quad h(\cdot;\alpha), \gamma \in \mathbb{R}^K, \quad \alpha \in \mathbb{R}^Q,$$

where α and γ denote the parameters of the base learners and $h(\cdot;\alpha) : \mathbb{R}^p \to \mathbb{R}^K$. For instance, this is the case for regression trees where α indicates the splitting variables and the split locations, γ contains the terminal node values, and $h(\cdot;\alpha)$ is a function that maps the covariate to terminal tree nodes. In this case, hybrid gradient-Newton boosting first learns α_m using a gradient boosting step given in (7), and then γ_m is learned using a Newton boosting step defined in (8). For the latter, an explicit generalized least squares solution is obtained as

$$\gamma_m = \left(h_{\alpha_m}^T {\Psi^{\dagger}}^{-1} h_{\alpha_m}\right)^{-1} h_{\alpha_m}^T {\Psi^{\dagger}}^{-1}(y - F_{m-1}),$$

where $h_{\alpha_m} \in \mathbb{R}^{n\times K}$ is the matrix with entries $(h_{\alpha_m})_{ik} = h(X_i;\alpha_m)_k$, $i = 1, \dots, n$, $k = 1, \dots, K$.

We note that in Newton boosting as well as hybrid gradient-Newton boosting, the norm of $f_m(\cdot)$, i.e. the step-length, does not depend on the scaling of the loss function, in particular not on σ^2. Since gradients are scale-dependent, this does not hold true for gradient boosting.

It has been empirically observed that damping the update in (6) results in higher predictive accuracy [Friedman, 2001]. This means that the update in (6) is damped by a factor ν:

$$F_m(\cdot) = F_{m-1}(\cdot) + \nu f_m(\cdot), \quad \nu > 0, \tag{9}$$

where ν is called the shrinkage parameter or learning rate.

As in finite dimensional optimization, functional gradient descent can be accelerated using momentum. For instance, Biau et al. [2019] and Lu et al. [2019] propose to use Nesterov acceleration [Nesterov, 2004] for gradient boosting.

3.2 Gaussian process boosting

A straightforward approach for minimizing (5) would consist of iteratively first doing one approximate functional gradient descent or Newton descent step for $F(\cdot)$ and then performing one gradient or (quasi-)Newton descent step for θ. Despite being attractive from a computational point of view, this has the following drawback. For finite samples, boosting tends to overfit, in particular for regression, and early stopping has to be applied as a form of regularization to prevent this. However, there is no guarantee that θ has converged to a minimum when early stopping is applied after a certain number of iterations. A possible solution to avoid this problem consists of doing coordinate descent, also called block descent, for both $F(\cdot)$ and θ, i.e. iteratively doing full optimization in both directions. However, this has the drawback that it is computationally expensive as both $F(\cdot)$ and the covariance parameters θ need to be repeatedly learned.

Our proposed solution presented in Algorithm 1 is to combine functional gradient or Newton boosting steps for $F(\cdot)$ with coordinate descent steps for θ. Specifically, in every iteration, we first determine $\theta_m = \text{argmin}_{\theta\in\Theta} L(y, F_{m-1}, \theta)$ and, based on this, update the ensemble of base learners using either a functional gradient descent step, a functional Newton step, or a combination of the two to obtain $F_m(\cdot) = F_{m-1}(\cdot) + \nu f_m(\cdot)$. We thus avoid the above-mentioned overfitting problem while being computationally more effective compared to doing coordinate descent in both directions.

Algorithm 1: GPBoost: Gaussian Process Boosting

```
input : Initial value $\theta_0 \in \Theta$, learning rate $\nu > 0$, number of boosting iterations
        $M \in \mathbb{N}$, BoostType $\in$ {"gradient", "newton", "hybrid"},
        NesterovAccel $\in$ {True, False}, and if NesterovAccel==True
        momentum sequence $\mu_m \in (0, 1]$
output: Mean function $\hat{F}(\cdot) = F_M(\cdot)$ and covariance parameters $\hat{\theta} = \theta_M$
 1: Initialize $F_0(\cdot) = \text{argmin}_{c\in\mathbb{R}} L(y, c \cdot 1, \theta_0)$
 2: for $m = 1$ to $M$ do
 3:   Find $\theta_m = \text{argmin}_{\theta\in\Theta} L(y, F_{m-1}, \theta)$ using a (accelerated) first- or second-order
      method for convex optimization initialized with $\theta_{m-1}$
 4:   if NesterovAccel==True then
 5:     Set $G_{m-1}(\cdot) = F_{m-1}(\cdot)$
 6:     if $m > 1$ then
 7:       Update $F_{m-1}(\cdot) = G_{m-1}(\cdot) + \mu_m(G_{m-1}(\cdot) - G_{m-2}(\cdot))$
 8:     end if
 9:   end if
10:   if BoostType=="gradient" then
11:     Find $f_m(\cdot) = \text{argmin}_{f(\cdot)\in\mathcal{S}} \left\|\Psi_m^{-1}(F_{m-1} - y) - f\right\|^2$
12:   else if BoostType=="newton" then
13:     Find $f_m(\cdot) = \text{argmin}_{f(\cdot)\in\mathcal{S}} (y - F_{m-1} - f)^T \Psi_m^{\dagger\,-1} (y - F_{m-1} - f)$
14:   else if BoostType=="hybrid" then
15:     Find $\alpha_m = \text{argmin}_{\alpha: f(\cdot)=h(\cdot;\alpha)^T\gamma\in\mathcal{S}} \left\|\Psi_m^{-1}(F_{m-1} - y) - f\right\|^2$
16:     Calculate $\gamma_m = \left(h_{\alpha_m}^T \Psi_m^{\dagger\,-1} h_{\alpha_m}\right)^{-1} h_{\alpha_m}^T \Psi_m^{\dagger\,-1}(y - F_{m-1})$
17:     Set $f_m(\cdot) = h(\cdot; \alpha_m)^T \gamma_m$
18:   end if
19:   Update $F_m(\cdot) = F_{m-1}(\cdot) + \nu f_m(\cdot)$
20: end for
```

We note that if the risk functional $R(F(\cdot), \theta) = L(y, F, \theta)\Big|_{F=F(X)}$ is convex in its two arguments $F(\cdot)$ and θ and Θ is a convex set, then (5) is a convex optimization problem since $\mathcal{H} = span(\mathcal{S})$ is also convex. Such an algorithm with gradient or Newton steps in $F(\cdot)$ and coordinate descent in θ thus converges to a minimizer of $R(F(\cdot), \theta)$ as long as the learning rate ν is not too large to avoid overshooting, i.e. that the risk increases when doing too large steps.

The coordinate descent step for finding $\theta_m = \text{argmin}_{\theta\in\Theta} L(y, F_{m-1}, \theta)$ is done using a first- or second-order method for convex optimization initialized with the covariance parameters θ_{m-1} of the previous iteration. In doing so, we avoid the full re-estimation of the covariance parameters in every boosting iteration. Examples of optimization algorithms for determining θ_m include gradient descent, gradient descent with Nesterov acceleration [Nesterov, 2004], and Fisher scoring which is a quasi Newton method that is often called "natural gradient descent" in machine learning [Amari, 1998]. For Gaussian processes, we have found in simulated experiments that when doing gradient descent, profiling out the error variance using the analytic formula in (11) increases convergence speed (results not tabulated). For Fisher scoring, on the other hand, profiling out the error variance can reduce convergence speed considerably (results not tabulated). Further, when doing gradient descent, the computational time to find minimizers can be reduced when adding adding Nesterov acceleration. For first- and second-order optimization methods, the gradient of $L(y, F, \theta)$ with respect to θ is required. This gradient is given by

$$\frac{\partial L(y, F, \theta)}{\partial \theta_k} = -\frac{1}{2}(y-F)^T\Psi^{-1}\frac{\partial \Psi}{\partial \theta_k}\Psi^{-1}(y-F) + \frac{1}{2}\text{tr}\left(\Psi^{-1}\frac{\partial \Psi}{\partial \theta_k}\right), \quad k = 1, \dots, q, \tag{10}$$

see, e.g., Williams and Rasmussen [2006], and the trace is calculated as

$$\text{tr}\left(\Psi^{-1}\frac{\partial \Psi}{\partial \theta_k}\right) = \sum_{i,j=1}^{n}\left(\Psi^{-1}\right)_{ij}\left(\frac{\partial \Psi}{\partial \theta_k}\right)_{ij}.$$

In addition, there is an explicit solution for the error variance parameter:

$$\sigma^2 = \frac{1}{n}\left(y - F_{m-1}\right)^T \Psi^{\dagger^{-1}}\left(y - F_{m-1}\right). \tag{11}$$

Further, the Fisher information matrix $I \in \mathbb{R}^{q\times q}$ for Fisher scoring, or natural gradient descent, is given by

$$(I)_{kl} = \frac{1}{2}\text{tr}\left(\Psi^{-1}\frac{\partial \Psi}{\partial \theta_k}\Psi^{-1}\frac{\partial \Psi}{\partial \theta_l}\right), \quad 1 \leq k, l \leq q.$$

In practice, we reparametrize all parameters θ_k with positivity constraints, such as marginal variance and range parameters, on the log-scale $\log(\theta_k)$ in order to constrain them to positive values during the numerical optimization.

Further, for the linear case, asymptotic theory [Stein, 1999] suggests that if the smallest eigenvalue of the Fisher information I tends to infinity as $n \to \infty$, we can expect that

$$I(\hat{\theta})^{1/2}(\hat{\theta} - \theta_0) \xrightarrow{d} \mathcal{N}(0, I),$$

where θ_0 denotes the population parameter and $I(\hat{\theta})^{1/2}$ is some matrix square root. Based on this, one can construct approximate confidence sets or intervals for θ.

Finally, note that in linear mixed effects models, i.e. if $F(x) = x^T\beta$, $L(y, F, \theta)$ is usually optimized by first profiling out the fixed effect part and then optimizing over θ using, e.g., a quasi-Newton method. In our case, this is not an option since there is no explicit solution for $F(\cdot)$ conditional on θ. Also note that restricted maximum likelihood (REML) estimation is often used for linear mixed effects models since otherwise covariance parameter estimates can be biased. This is, however, not possible in our case. We conjecture that early stopping has a similar effect as it prevents overfitting.

3.3 Out-of-sample learning for covariance parameters

It has recently been observed that state-of-the-art machine learning techniques such as neural networks, kernel machines, or boosting achieve a zero training loss while at the same time having excellent generalization properties [Zhang et al., 2017, Wyner et al., 2017, Belkin et al., 2018]. While many of these results have been found for classifiers, Belkin et al. [2019], for instance, present regression examples where interpolating learners have low generalization error. Further, Wyner et al. [2017, Section 3.2] provide an intuitive argument for why certain interpolation regression learners can generalize well to novel data.

In line with this, we find in our simulated experiments in Section 4 that estimates of the error variance σ^2 are often too small also when doing early stopping by monitoring a test error. A way to alleviate this bias problem of the error variance is to estimate the covariance parameters using out-of-sample data, i.e. using a validation data set or by doing cross-validation. To avoid that the mean function and/or the covariance parameters θ are only learned on a fraction of the full data, we propose a two-step approach presented in the GPBoostOOS Algorithm 2. In brief, the GPBoostOOS algorithm first runs the GPBoost algorithm and obtains predictions for the mean function on the validation data. The covariance parameters are then estimated on the validation data using the predicted mean function. Finally, the GPBoost algorithm is run a second time without estimating any covariance parameters, though. Note that when k-fold cross-validation is used, both the mean function and the covariance parameter are learned using the full data.

Algorithm 2: GPBoostOOS: Gaussian Process Boosting with Out-Of-Sample covariance parameter estimation

```
input : Initial value $\theta_0 \in \Theta$, learning rate $\nu > 0$, number of boosting iterations
        $M \in \mathbb{N}$, BoostType $\in$ {"gradient", "newton", "hybrid"},
        NesterovAccel $\in$ {True, False}, and if NesterovAccel==True
        momentum sequence $\mu_m \in (0, 1]$
output: Mean function $\hat{F}(\cdot)$ and covariance parameters $\hat{\theta}$
  1: Partition the data into training and validation sets, e.g. using $k$-fold cross-validation
  2: Run the GPBoost algorithm on the training data and generate predictions for the
     mean function on the validation data $\hat{F}_{val}$
  3: Find $\hat{\theta} = \text{argmin}_{\theta \in \Theta} L(y_{val}, \hat{F}_{val}, \theta)$ using the validation data with response $y_{val}$
  4: Run the GPBoost algorithm on the full data while holding the covariance parameters
     $\theta$ fixed at $\hat{\theta}$, i.e. skipping line 3, to obtain $\hat{F}(\cdot)$
```

3.4 Efficient learning for large data

For efficient learning of trees, several approaches exist for scaling the computations to large data [Chen and Guestrin, 2016, Ke et al., 2017, Prokhorenkova et al., 2018]. In this article, we use the approach presented in Ke et al. [2017].

Concerning covariance parameters, we adopt the following solutions for reducing computational costs. If the random effects b consists of only grouped random effects, Ψ is sparse and computations can be done efficiently using sparse matrix calculations. Further, we use the Sherman-Morrison-Woodbury formula

$$\left(Z\Sigma Z^T + \sigma^2 I_n\right)^{-1} = \sigma^{-2} I_n - \sigma^{-2} Z \left(\sigma^2 \Sigma^{-1} + Z^T Z\right)^{-1} Z^T \tag{12}$$

for calculating gradients with respect to F and θ and for evaluating the log-likelihood since the dimension of the random effects m is typically smaller than the number of samples n if there are only grouped random effects.

If b contains a Gaussian process with a non-sparse covariance matrix, both the computational cost and the required memory do not scale well in the number of observed locations. In this case, one has to rely on some approximation to make calculations feasible. We choose to use a Vecchia approximation [Vecchia, 1988, Datta et al., 2016, Katzfuss and Guinness, 2017, Finley et al., 2019], also denoted as nearest-neighbor Gaussian process (NNGP) model by Datta et al. [2016], as it has several advantages over other Gaussian process approximations for large data [Guinness, 2018]. Roughly speaking, the idea of the Vecchia approximation is to approximate a Cholesky factor of the precision matrix using a sparse matrix and thus also obtain a sparse approximate precision matrix. In the following, we briefly review how this is obtained in our case and then show how gradients of the negative log-likelihood given in (10) can be calculated efficiently. To the best of our knowledge, the latter result is novel.

3.4.1 Vecchia approximation for the response variable y

Vecchia approximations are a special form of composite likelihood methods [Varin et al., 2011]. In our case, the likelihood $p(y|F,\theta)$ is approximated as

$$\begin{aligned} p(y|F,\theta) &= \prod_{i=1}^{n} p(y_i|(y_1,\dots,y_{i-1}),F,\theta) \\ &\approx \prod_{i=1}^{n} p(y_i|y_{N(i)},F,\theta), \end{aligned} \tag{13}$$

where $y_{N(i)}$ are subsets of the conditioning sets $(y_1,\dots,y_{i-1})$ and $N(i)$ denote the corresponding subsets of indices. As is commonly done, we choose $N(i)$ as the indices of the m nearest neighbors of s_i among $s_1,\dots,s_{i-1}$ if $i>m+1$ and, in the case $i\leq m+1$, $N(i)$ equals $(1,\dots,i-1)$.

By standard arguments for conditional Gaussian distributions, we have

$$p(y_i|y_{N(i)},F,\theta) = \mathcal{N}\left(y_i \mid F_i + A_i\left(y_{N(i)} - F_{N(i)}\right), D_i\right),$$

where $\mathcal{N}(x|\mu,\Xi)$ denotes the normal density with mean vector μ and covariance matrix Ξ evaluated at x, and $A_i \in \mathbb{R}^{1\times|N(i)|}$ and $D_i\in\mathbb{R}$, where $|N(i)|$ denotes the size of the set $N(i)$, are given by

$$\begin{aligned} A_i &= \left(Z\Sigma Z^T\right)_{i,N(i)}\left(\left(Z\Sigma Z^T+\sigma^2 I_n\right)_{N(i)}\right)^{-1} \\ D_i &= \left(Z\Sigma Z^T+\sigma^2 I_n\right)_{i,i} - A_i\left(Z\Sigma Z^T\right)_{N(i),i} \end{aligned} \tag{14}$$

where $\Sigma = (c(s_l,s_k))_{l,k}$, $1\leq l,k\leq n$ is the covariance matrix of b, $c(\cdot,\cdot)$ the covariance function, $M_{i,N(i)}$ denotes the sub-matrix of M consisting of row i and columns $N(i)$, and $M_{N(i)}$ denotes the sub-matrix of a matrix M consisting of rows $N(i)$ and columns $N(i)$. Note that if Z is a diagonal matrix, we have

$$\left(Z\Sigma Z^T\right)_{N(i)} = Z_{N(i)}\Sigma_{N(i)}Z_{N(i)} = \Sigma_{N(i)}\odot\left(zz^T\right)_{N(i)},$$

where z is the diagonal of Z, i.e., the covariate data, and $\odot$ denotes the Hadamard product. The use of this relationship can lead to a reduction in computation cost, in particular for random coefficient models; see Dambon et al. [2021].

We further denote by B the lower triangular matrix with 1's on the diagonal, off-diagonal entries

$$(B)_{i,N(i)} = -A_i, \tag{15}$$

and 0's otherwise, and by D a diagonal matrix with D_i on the diagonal. We then obtain the following approximate distribution

$$y \overset{approx}{\sim} \mathcal{N}\left(F(X),\tilde{\Psi}\right), \qquad \tilde{\Psi} = B^{-1}DB^{-T}, \tag{16}$$

and the corresponding precision matrix is given by

$$\tilde{\Psi}^{-1} = B^T D^{-1} B, \tag{17}$$

where the Cholesky factor B and also $\tilde{\Psi}^{-1}$ are sparse.

3.4.2 Efficient calculation of the gradient and Fisher information for the Vecchia approximation

In the following, we show how the gradient and the Fisher information of the approximate log-likelihood of the Vecchia approximation given in (16) can be calculated efficiently. To the best of our knowledge, the following results are novel. Guinness [2019] also presents a way for computing the gradient and Fisher information for the Vecchia approximation. However, he uses a different representation of the approximate likelihood by writing conditional densities in (13) as ratios of joint and marginal densities and thus obtains different way for calculating the gradient and Fisher information compare to our result. Guinness [2019] motivates his approach by claiming that for calculating the gradient in (10), "[n]ot only is $\left[\frac{\partial\Psi}{\partial\theta_k}\right]$ too large to store in memory, the covariances [Ψ] are not easily computable, nor are their partial derivatives". The following result in Proposition 3.1 shows that that this is not true, i.e. $\frac{\partial\Psi}{\partial\theta_k}$ does not need to be stored in memory and neither Ψ nor its partial derivatives need to be computed.

Proposition 3.1. *The gradient of the negative log-likelihood $\tilde{L}(y, F, \theta)$ for the Vecchia approximation given in* (16) *can be calculated as*

$$\frac{\partial\tilde{L}(y,F,\theta)}{\partial\theta_k} = \frac{1}{2\sigma^2}\left(2u_k^T u - u^T\frac{\partial D}{\partial\theta_k}u\right) + \frac{1}{2}\sum_{i=1}^{n}\frac{1}{D_i}\frac{\partial D_i}{\partial\theta_k}, \quad 1\le k\le q,$$

where

$$u = D^{-1}B(y-F) \quad \text{and} \quad u_k = \frac{\partial B}{\partial\theta_k}(y-F), \tag{18}$$

and $\frac{\partial B}{\partial\theta_k}$ are lower triangular and $\frac{\partial D}{\partial\theta_k}$ diagonal matrices with non-zero entries given by

$$\begin{aligned}
\left(\frac{\partial B}{\partial\theta_k}\right)_{i,N(i)} =& -\frac{\partial A_i}{\partial\theta_k}\\
=& -\left(Z\frac{\partial\Sigma}{\partial\theta_k}Z^T\right)_{i,N(i)}\left(\left(Z\Sigma Z^T+\sigma^2 I_n\right)_{N(i)}\right)^{-1}\\
&+\left(Z\Sigma Z^T\right)_{i,N(i)}\left(\left(Z\Sigma Z^T+\sigma^2 I_n\right)_{N(i)}\right)^{-1}\left(Z\frac{\partial\Sigma}{\partial\theta_k}Z^T\right)_{N(i)}\left(\left(Z\Sigma Z^T+\sigma^2 I_n\right)_{N(i)}\right)^{-1},\\
\frac{\partial D_i}{\partial\theta_k} =& \left(Z\frac{\partial\Sigma}{\partial\theta_k}Z^T\right)_{i,i} - \frac{\partial A_i}{\partial\theta_k}\left(Z^T\Sigma Z\right)_{N(i),i} - A_i\left(Z^T\frac{\partial\Sigma}{\partial\theta_k}Z\right)_{N(i),i},
\end{aligned}$$

for $1 < k \le q$, and for $k = 1$, the non-zero entries of $\frac{\partial B}{\partial\theta_k}$ and $\frac{\partial D}{\partial\theta_k}$ are

$$\begin{aligned}
\left(\frac{\partial B}{\partial\sigma^2}\right)_{i,N(i)} =& \left(Z\Sigma Z^T\right)_{i,N(i)}\left(\left(Z\Sigma Z^T+\sigma^2 I_n\right)_{N(i)}\right)^{-2},\\
\frac{\partial D_i}{\partial\sigma^2} =& 1 - \frac{\partial A_i}{\partial\sigma^2}\left(Z^T\Sigma Z\right)_{N(i),i}.
\end{aligned}$$

A proof can be found in Appendix B. Further, the Fisher information for the Vecchia approximation in (16) can be calculated using the following result.

Proposition 3.2. *The Fisher information for the Vecchia approximation matrix in* (16) *has entries*

$$(I)_{kl} = \sum_{i,j=1}^{n}\left(D^{-1}\frac{\partial B}{\partial\theta_k}B^{-1}\right)_{ij}\left(\frac{\partial B}{\partial\theta_l}B^{-1}D\right)_{ij} + \frac{1}{2}\sum_{i=1}^{n}D_i{}^{-2}\frac{\partial D_i}{\partial\theta_k}\frac{\partial D_i}{\partial\theta_l}, \quad 1\le k\le q, \tag{19}$$

where $\frac{\partial B}{\partial\theta_k}$ and $\frac{\partial D}{\partial\theta_k}$ are lower triangular and diagonal matrices defined in Proposition 3.1

A proof can be found in Appendix B. This Fisher information can be used for finding a maximum of the (approximate) likelihood using the Fisher scoring algorithm. Concerning an approximate covariance matrix for $\hat{\theta}$ using the asymptotic result mentioned in Section 3.2, we note that since the Vecchia approximation results in a misspecified model, the Fisher information matrix needs to be replaced by the Godambe information matrix [Godambe, 1960] $G = HI^{-1}H$, where H is the negative expected Hessian of the log-likelihood.

3.5 Prediction

Let $y_p \in \mathbb{R}^{n_p}$ denote the random variable for which predictions should be made. We have

$$\begin{aligned}\begin{pmatrix} y \\ y_p \end{pmatrix} &= \begin{pmatrix} F(X) \\ F(X_p) \end{pmatrix} + \begin{pmatrix} (Z, 0_{n\times m_p}) \\ Z_p \end{pmatrix} \begin{pmatrix} b \\ b_p \end{pmatrix} + \begin{pmatrix} \epsilon \\ \epsilon_p \end{pmatrix}, \\ &\sim \mathcal{N}\left(\begin{pmatrix} F(X) \\ F(X_p) \end{pmatrix}, \begin{pmatrix} Z\Sigma Z^T + \sigma^2 I_n & Z(\Sigma, \Sigma_{op}) Z_p^T \\ Z_p(\Sigma, \Sigma_{op})^T Z^T & Z_p \begin{pmatrix} \Sigma & \Sigma_{op} \\ \Sigma_{op}^T & \Sigma_p \end{pmatrix} Z_p^T + \sigma^2 I_{n_p} \end{pmatrix} \right)\end{aligned} \tag{20}$$

where $b_p \in \mathbb{R}^{m_p}$ is a vector of m_p random effects, for which no data has been observed in y, $(Z, 0_{n\times m_p}) \in \mathbb{R}^{n\times(m+m_p)}$, $0_{n\times m_p} \in \mathbb{R}^{n\times m_p}$ is a matrix of zeros, the matrix $Z_p \in \mathbb{R}^{n_p\times(m+m_p)}$ relates the vector of observed and new random effects $(b^T, b_p^T)^T \in \mathbb{R}^{m+m_p}$ to the prediction variable y_p, $(\Sigma, \Sigma_{op}) \in \mathbb{R}^{m\times(m+m_p)}$, $\Sigma_{op} = \text{Cov}(b, b_p)$, $\Sigma_p = \text{Cov}(b_p)$, and $X_p \in \mathbb{R}^{n_p\times p}$ is the predictor variable matrix of the predictions.

Note that the prediction random variable y_p can be related to both existing random effects b, for which data y has been observed, and also new, unobserved random effects b_p. This distinction is particularly relevant for grouped random effects, where predictions can be made for samples of groups for which data has already been observed in y, or also for new groups for which no data has been observed in y. Further, this can also be useful for Gaussian process models for distinguishing when predictions should be made for y_p, or b_p, at new locations, or when b should be predicted at observed locations.

From (20), it follows that the conditional distribution $y_p|y$ is given by

$$y_p|y \sim \mathcal{N}\left(\mu_p, \Xi_p\right),$$

where

$$\begin{aligned}\mu_p =& F(X_p) + Z_p(\Sigma, \Sigma_{op})^T Z^T \left(Z\Sigma Z^T + \sigma^2 I_n\right)^{-1} (y - F(X)) \\ \Xi_p =& Z_p \begin{pmatrix} \Sigma & \Sigma_{op} \\ \Sigma_{op}^T & \Sigma_p \end{pmatrix} Z_p^T + \sigma^2 I_{n_p} - Z_p(\Sigma, \Sigma_{op})^T Z^T \left(Z\Sigma Z^T + \sigma^2 I_n\right)^{-1} Z(\Sigma, \Sigma_{op}) Z_p^T.\end{aligned} \tag{21}$$

Depending on the application, i.e. if $n \gg m$, the above quantities can be more efficiently calculated using the Sherman-Morrison-Woodbury formula given in (12). Further, predictions for the latent b, b_p, or $F(X_p)+Z_p\begin{pmatrix} b \\ b_p \end{pmatrix}$ can be done analogously with minor modifications, e.g., dropping the error variance term $\sigma^2 I_{n_p}$ from the covariance matrix in (21).

3.5.1 Prediction using the Vecchia approximation

Similarly as for parameter estimation, Vecchia approximations can also be used for making predictions. Specifically, predictions can be obtained by applying a Vecchia approximation to the joint response vector at observed and prediction locations. When doing so, one has to choose an ordering among the joint set of observed and predicted locations. We assume that either the observed or the predicted locations appear first in the ordering of the response variable. The former has the advantage that the nearest neighbors found for estimation can be reused and that the predictive distributions have the simple form given below in (22). On the other hand, if prediction locations appear first in the ordering, the approximations of predictive distributions are generally more accurate. See Katzfuss et al. [2018] for a comparison of different approaches for making predictions with Vecchia approximations.

Proposition 3.3. *Assume that prediction are made at n_p locations $s_{p,1}, \dots, s_{p,n_p}$ with covariate data X_p. When applying the Vecchia approximation in (16) to the response vector $(y, y_p)^T$ with the observed response y appearing first in the ordering, the conditional distribution $y_p|y$ is given by*

$$y_p|y \sim \mathcal{N}\left(\mu_p, \Xi_p\right),$$

where

$$\begin{aligned}\mu_p =& F(X_p) - B_p^{-1} B_{po} \left(y - F(X)\right) \\ \Xi_p =& B_p^{-1} D_p B_p^{-T},\end{aligned} \tag{22}$$

and $B_{po} \in \mathbb{R}^{n_p \times n}$, $B_p \in \mathbb{R}^{n_p \times n_p}$, ${D_p}^{-1} \in \mathbb{R}^{n_p \times n_p}$ are the following submatrices of the Vecchia approximated precision matrix $\tilde{Cov}\left((y, y_p)^T\right)^{-1}$:

$$\tilde{Cov}\left((y, y_p)^T\right)^{-1} = \begin{pmatrix} B & 0 \\ B_{po} & B_p \end{pmatrix}^T \begin{pmatrix} D^{-1} & 0 \\ 0 & {D_p}^{-1} \end{pmatrix} \begin{pmatrix} B & 0 \\ B_{po} & B_p \end{pmatrix}, \tag{23}$$

and B and D are defined in (14) *and* (15)*.*

A proof can be found in Appendix B. Note that D_p is a diagonal matrix, B_p is a lower triangular matrix with 1's on the diagonal and non-zero off-diagonal entries corresponding to the nearest neighbors of the prediction locations among the prediction locations themselves $s_{p,1}, \dots, s_{p,n_p}$, and B_{po} has non-zero entries corresponding to the nearest neighbors of the prediction locations among the observed locations $s_1, \dots, s_n$.

If only univariate predictive distributions are of interest, computational costs can be additionally reduced when restricting that one conditions on observed locations only in (13). The latter means that for every prediction location $s_{p,i}$, one conditions only on observed data $y_{N(i)}$ where $N(i)$ denotes the set of nearest neighbors for location $s_{p,i}$. In this case, B_p is an identity matrix and the predictive covariance matrix Ξ_p is a diagonal matrix. The latter can be a drawback if multivariate predictive distributions are of interest.

Proposition 3.4. *Assume that prediction are made at n_p locations $s_{p,1}, \dots, s_{p,n_p}$ with covariate data X_p. When applying the Vecchia approximation in* (16) *to the response vector $(y_p, y)^T$ with the predicted response y_p appearing first in the ordering, the conditional distribution $y_p|y$ is given by*

$$y_p|y \sim \mathcal{N}\left(\mu_p, \Xi_p\right),$$

with

$$\begin{aligned}\mu_p =& F(X_p) - \left(B_p^T {D_p}^{-1} B_p + B_{op}^T {D_o}^{-1} B_{op}\right)^{-1} B_{op}^T {D_o}^{-1} B_o \left(y - F(X)\right) \\ \Xi_p =& \left(B_p^T {D_p}^{-1} B_p + B_{op}^T {D_o}^{-1} B_{op}\right)^{-1},\end{aligned} \tag{24}$$

where $B_o, D_o \in \mathbb{R}^{n \times n}$, $B_{op} \in \mathbb{R}^{n \times n_p}$, $B_p, D_p \in \mathbb{R}^{n_p \times n_p}$, ${D_p}^{-1} \in \mathbb{R}^{n_p \times n_p}$ are the following submatrices of the Vecchia approximated precision matrix $\tilde{Cov}\left((y_p, y)^T\right)^{-1}$:

$$\tilde{Cov}\left((y_p, y)^T\right)^{-1} = \begin{pmatrix} B_p & 0 \\ B_{op} & B_o \end{pmatrix}^T \begin{pmatrix} {D_p}^{-1} & 0 \\ 0 & {D_o}^{-1} \end{pmatrix} \begin{pmatrix} B_p & 0 \\ B_{op} & B_o \end{pmatrix}.$$

A proof can be found in Appendix B.

3.6 Software implementation

The GPBoost algorithm is implemented in the `gpboost` library written in C++ with a C application programming interface (API) and corresponding Python and R packages. See `https://github.com/fabsig/GPBoost` for more information. For linear algebra calculations, we rely on the `Eigen` library [Guennebaud et al., 2010]. Sparse matrix algebra is used, in particular for calculating Cholesky decompositions, whenever covariance matrices are sparse, e.g. in the case of grouped random effects. In addition, to speed up computations for solving sparse linear triangular equation systems where the right-hand side is also sparse, we use the function `cs_spsolve` from the `CSparse` library [Davis, 2005] where the non-zero entries of the solutions are determined using a depth-first search algorithm. Further, multi-processor parallelization is done using `OpenMP`. For the tree-boosting part, in particular the tree growing algorithm, we use the `LightGBM` library [Ke et al., 2017].

4 Simulated experiments

In the following, we use simulation to investigate the predictive accuracy as well as the properties of the covariances parameter and mean function estimates of the GPBoost algorithm.

4.1 Simulation setting and evaluation criteria

For the random effects term Zb, we consider grouped random effects with a single level of grouping and a spatial Gaussian process model with an exponential covariance function

$$c(s, s') = \sigma_1^2 \exp(-\|s - s'\|/\rho), \tag{25}$$

where the locations s are in $[0,1]^2$ and $\rho = 0.1$. The marginal variance in both models is set to $\sigma_1^2 = 1$ and the error variance equals $\sigma^2 = 1$ such that we have a signal-to-noise ratio of 1 between the random effect and the error term. Concerning the mean function $F(\cdot)$ and the predictor variables X, we consider the following different specifications:

$$\begin{aligned}
F(x) &= C \cdot \tan^{-1}\left(\frac{x_2 x_3 - 1 - \frac{1}{x_2 x_4}}{x_1}\right), \quad x = (x_1, x_2, x_3, x_4)^T, \qquad \text{(‘friedman3’)}, \\
&\qquad x_1 \sim Unif(0, 100), x_2 \sim Unif(40\pi, 560\pi), x_3 \sim Unif(0, 1), x_4 \sim Unif(1, 11), \\
F(x) &= C \cdot (2x_1 + x_2^2 + 4 \cdot 1_{\{x_3 > 0\}} + 2\log(|x_1|)x_3), \quad x = (x_1, \dots, x_9)^T, \qquad \text{(‘hajjem’)}, \\
&\qquad x \sim \mathcal{N}(0, I_9), \\
F(x) &= C \cdot (1 + x_1 + x_2), \quad x = (x_1, x_2)^T, \qquad \text{(‘linear’)}, \\
&\qquad x_1, x_2 \overset{iid}{\sim} Unif(0, 1).
\end{aligned} \tag{26}$$

The function ‘friedman3’ was first used in Friedman [1991] and has since then often been used to compare non-parametric regression models, and the function ‘hajjem’ has been used in Hajjem et al. [2014] to compare non-parametric mixed effects models. We also include a linear function in order to investigate how our approach compares to a linear mixed effects model when the data generating process is linear. The constant C is chosen such that the variance of $F(x)$ equals approximately 1, i.e. that $Fx)$ has the same signal strength as the random effects.

We simulate 100 times training data sets of size n and two test data sets each also of size n using the specification in (1). All models are trained on the training data and evaluated on the test data. In every simulation run, the two test data sets, denoted briefly as “interpolation” and “extrapolation” test sets, are generated as follows. For the grouped random effects model, the “interpolation” test data set consists of random effects for the same groups as in the training data, and the “extrapolation” test data contains m independent random effects for new groups that have not been observed in the training data. For the Gaussian process model, training data locations are samples uniformly from $[0,1]^2$ excluding $[0.5,1]^2$, and the “interpolation” test data set is obtained by also simulating locations uniformly in the same area. Further, the “extrapolation” test data contains locations sampled uniformly from $[0.5,1]^2$. I.e., predictions for the “extrapolation” test data are to some degree extrapolations. Figure 1 illustrates this. The mean functions of all data sets are simulated independently using the specification in (26).

We use a sample size of $n = 5000$ for the grouped random effects with $m = 500$ different groups. For the Gaussian process model, we use a sample size of $n = 500$. The reason for using a smaller sample size is that this allows us to do all calculations exactly and avoid any approximation error due to a large data approximation. In Section 3, we show how learning can be done for large data, and we use this in the application in Section 5.

Concerning predictive accuracy, we measure both the accuracy of point predictions and probabilistic predictions. Probabilistic predictions are obtained for mixed effects models as outlined in Section 3.5. For the approaches that do not rely on a probabilistic model for the random effects (mboost and LSBoost), we estimate the variance of the residuals on the training data, and use

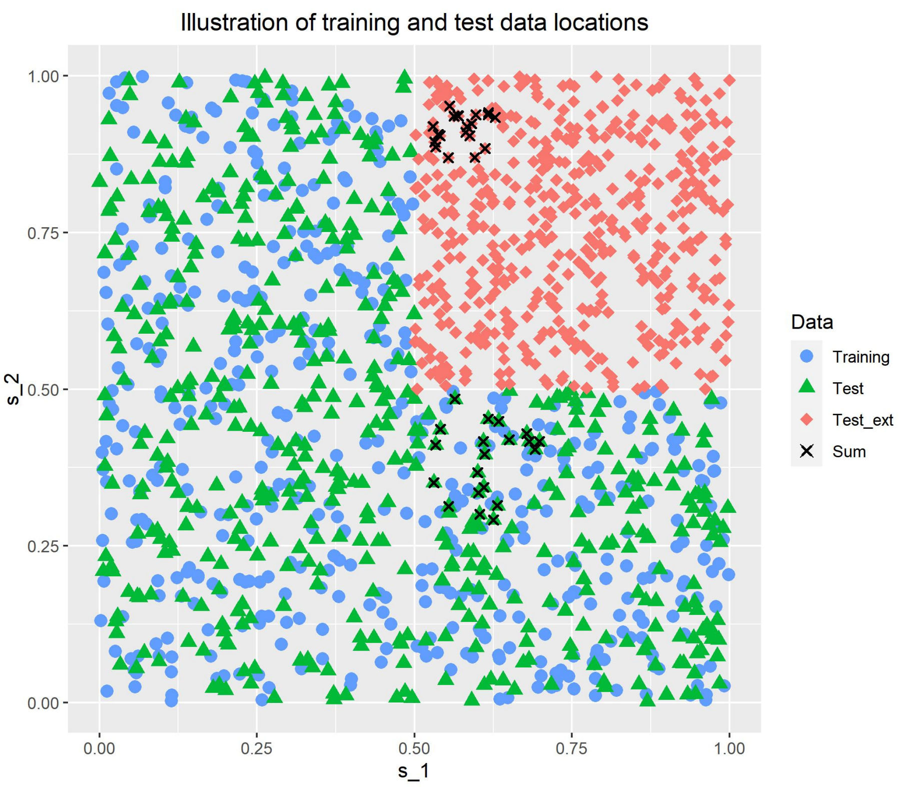

Figure 1: Example of locations for training and test data for the Gaussian process model. "Test" and "Test_ext" refers to locations of the "interpolation and "extrapolation" test data sets, respectively. The black crosses show examples of locations for which predictions of sums are made.

this for obtaining independent Gaussian predictive distributions. Point predictions are evaluated using the the root mean square error (RMSE) and probabilistic predictions are evaluated using the continuous ranked probability score (CRPS) [Gneiting et al., 2007].

Additionally, we also evaluate the accuracy when predicting sums of $n' = 20$ observations $y_p = (y_{p,1}, \dots, y_{p,n'})^T$. For spatial data, for instance, such predictions are required in meteorology for predicting the total precipitation over a catchment area or in real estate for predicting the total value of a portfolio of objects. Note that if a multivariate predictive distribution for y_p is given by $y_p|y \sim \mathcal{N}(\mu_p, \Xi_p)$, then the predictive distribution of the sum is obtained as

$$\mathbf{1}^T y_p|y \sim \mathcal{N}\left(\mathbf{1}^T \mu_p, \mathbf{1}^T \Xi_p \mathbf{1}\right),$$

where $\mathbf{1}$ is a vector of ones $\mathbf{1} = (1, \dots, 1)^T$. In total, we predict 50 times sums of different samples y_p in every simulation run. Half of these samples are from the "interpolation" and the "extrapolation" test data sets, respectively. Specifically, for the spatial data, we randomly select 25 times disjoints sets of 20 observations which are close together in space for both test sets. We obtain these sets of 20 observations by randomly selecting a location and then determining its 19 nearest neighbors and then iteratively continuing in the same manner with the remaining locations. This is illustrated in Figure 1. For the grouped random effects, the 20 observations simply consist of all samples of two groups each of size 10.

In addition, we also evaluate the accuracy to learn the mean function F and the random effects b. For doing this, we only use the "interpolation" test data sets; see above and Figure 1. Further, we also consider the accuracy of estimates for the variance and covariance parameters θ. We note that for the boosting approaches which model the spatial or grouped random effect using base learners

(LSBoost and mboost), no estimates for covariance parameters, the mean function F, and the random effects b can be obtained. The exception are grouped random effects where mboost reports estimated random effects b. This then also allows for obtaining an estimate for F by assigning the test data a certain group from the training data and subtracting the value of the estimated random effect for this group from the obtained predictions. In the same way, we also obtain predictions for grouped data of unobserved groups for mboost.

4.2 Methods considered and choice of tuning parameters

We compare our proposed methodology with the following alternative approaches: linear mixed effects models, model-based boosting ('mboost') [Hothorn et al., 2010], mixed-effects random forest ('MERF') [Hajjem et al., 2014], RE-EM trees ('REEMtree') [Sela and Simonoff, 2012], and independent gradient boosting with a square loss ('LSBoost'). In addition to the GPBoost algorithm, we also consider the GPBoostOOS algorithm where covariance parameters are estimated using 4-fold cross-validation. The MERF and REEMtree algorithms can only be used for the grouped random effects and not for the Gaussian process models. For all boosting algorithms, we use gradient boosting without Nesterov acceleration and trees as base learners, except for the grouped and spatial random effects in mboost where splines and Ridge regression are used. In independent gradient boosting with a square loss, the locations of the Gaussian process and the categorical grouping variable for the grouped random effects are included as additional predictor variables in the mean function F, and samples are assumed to be independent conditional on F.

Learning and prediction with the GPBoost and GPBoostOOS algorithms, the linear models, and gradient boosting with a square loss ('LSBoost') is done using the `GPBoost` library version 0.3.0 compiled with the MSVC compiler version 19.24.28315.0 and OpenMP version 2.0.[3] For the linear mixed effects model, the GPBoost algorithm, and the GPBoostOOS algorithm, optima for covariance parameters θ are found using Nesterov accelerated gradient descent. Further, for boosting with a square loss applied to the grouped random effects data, we consider the grouping variable as a numeric variable and not as a categorical variable as suggested by the authors of `LightGBM`[4] since the number of categories is large. The results are slightly worse when considering the grouping variable as a categorical variable (results not tabulated). Concerning the mboost algorithm, we use the `mboost` R package [Hofner et al., 2014] version 2.9-2, where spatial effects are modeled using bivariate P-spline base learner (`bspatial` with `df=6`), grouped random effects are modeled using random effects base learners (`brandom` with `df=4`), and all other predictor variables are modeled using trees as base learners. For the MERF algorithm, we use the `merf` Python package version 0.3, and for the REEMtree algorithm, we use the `REEMtree` R package version 0.90.3. The number of iterations of the MERF algorithm is set to 100. Increasing this value does not change our findings (results not tabulated). However, we note that we often do not observe convergence of the MERF algorithm, no matter how long we let it run.[5] This might be related to the fact that the MERF algorithm is not a properly defined EM algorithm; see Section 1.1. All calculations are done on a laptop with a 2.9 GHz quad-core processor and 16 GB of random-access memory (RAM).

Tuning parameters are chosen by simulating 10 additional training and test sets, and choosing the values from a grid that minimize the average RMSE on the test sets. In doing so, we use the union of both the "interpolation" and "extrapolation" test data sets, see Section 4 above, to calculate the RMSEs for tuning parameter selection. For all boosting methods, we consider the

[3]When using another compiler such as a GNU compiler which supports a higher version of OpenMP, computations for the random effects part can become considerably faster (e.g. more than twice as fast for the GP model when using OpenMP 4.5). However, the tree-boosting part of `LightGBM` is slower when using a GNU compiler.

[4]`https://lightgbm.readthedocs.io/en/latest/Advanced-Topics.html#categorical-feature-support` (retrieved on December 10, 2020)

[5]This is an observation that has also been made by the creator of the `MERF` package in a blog post; see https://towardsdatascience.com/mixed-effects-random-forests-6ecbb85cb177 (retrieved on December 10, 2020).

following grid of tuning parameters: the number of boosting iterations $M \in \{1, \dots, 1000\}$, the learning rate $\nu \in \{0.1, 0.05, 0.01\}$, the maximal tree depth $\in \{1, 5, 10\}$, and the minimal number of samples per leaf $\in \{1, 10, 100\}$. To investigate the impact of the choice of tuning parameters, we also consider the GPBoost algorithm where only the number of boosting iterations is selected on the validation sets and the learning rate is held fixed at $\nu = 0.1$, the maximal tree depth is set to 5, and the minimal number of samples per leaf is 10. We refer to this as 'GPBFixLR' in the results below. For random forests, the choice of the tuning parameters is less important compared to boosting. For the MERF algorithm, we choose the proportion of variables considered for making splits $\in \{0.5, 0.75, 1\}$. We do not impose a maximal tree depth limit and set the number of trees to 300. These are the default values of the `MERF` package and have also been used in Hajjem et al. [2014]. Note that the MERF algorithm implemented in the `MERF` package is very slow, see the results below, and parameter tuning is thus computationally demanding. For the `REEMtree` package, which relies on the `rpart` R package, trees are cost-complexity pruned and the amount of pruning is chosen using 10-fold cross-validation on the training data.

4.3 Results

The results for the 'friedman3' mean function are reported in Tables 1 and 2 and Figures 2 and 3 for the grouped random effects and the spatial Gaussian process models, respectively. The results for the other two fixed effects functions are reported in Appendix C. In the tables, we report average values of the predictive accuracy metrics as well as standard deviations over the simulation runs. Further, we calculate p-values of paired t-tests comparing the GPBoost algorithm to the other approaches. Specifically, for every evaluation criterion, we test the null hypothesis of equality between the accuracy measure of GPBoost algorithm and every alternative approach using the paired samples from the different simulation runs. For the covariance parameters, we report RMSEs over the simulation runs. In addition, Figures 2 and 3 summarize the results in a graphical way using violin plots.

Considering the results for the grouped random effects reported in Table 1 and Figure 2, we find that the GPBoost algorithm significantly outperforms all other methods in all predictive accuracy measures including the ones concerning the learning, or prediction, of both the mean function $F(\cdot)$ and the random effects b. The same holds also true for the other non-linear mean function ('hajjem') whose results are reported in Table 5 in the appendix. Not surprisingly, a linear mixed effects model (LinearME) performs considerably worse than the GPBoost algorithms in all predictive accuracy measures and also in the estimation of the variance parameters θ and the mean function $F(\cdot)$. Gradient boosting with a square loss including the grouping variable as a categorical variable (LSBoost) also has significantly lower predictive accuracy in all metrics. However, the difference in predictive accuracy between the GPBoost and the LSBoost algorithm is particularly large on the "interpolation" test data sets (see 'RMSE' and 'CRPS'), which consists of observations for groups already observed in the training data, and the difference is relatively smaller on the "extrapolation" test sets containing new groups (see 'RMSE_ext' and 'CRPS_ext'). The likely reason for this is that when including the grouping variable as a categorical variable, least squares gradient boosting has difficulty in learning the grouped effects, i.e. it has large variance, as the number of categories is large in relation to the total sample size. The situation is similar for the mboost algorithm. Here, we can obtain estimates for the random effects b, and we observe that the accuracy of these estimates is very low (see 'RMSE_b'). Concerning the "interpolation" test data sets, the MERF algorithm has the second highest predictive accuracy (see 'RMSE' and 'CRPS'). However, for the prediction of observations of new groups and, in particular, the learning of the mean function $F(\cdot)$, the MERF algorithm performs worse (see 'RMSE_ext', 'CRPS_ext', and 'RMSE_F'). In summary, both boosting approaches which model the random effects using deterministic base learners and assume independent observations (LSBoost and mboost) have particular difficulty learning the random effects b, and the MERF algorithm produces inaccurate estimates for the mean function $F(\cdot)$.

Concerning variance parameters estimates, we observe no major differences among the methods in the RMSE of the variance of the random effects ('sigma2_b'), except for the linear mixed

	GPBoost	LinearME	LSBoost	mboost	MERF	REEMtree	GPBoostOOS	GPBFixLR
RMSE	**1.065**	1.237	1.416	1.239	1.095	1.12	1.065	1.065
(sd)	(0.0116)	(0.0124)	(0.0248)	(0.0167)	(0.0117)	(0.0133)	(0.0116)	(0.0117)
[p-val]		[2.65e-121]	[2.01e-115]	[2.85e-107]	[4.38e-90]	[7.13e-87]	[0.246]	[2.91e-10]
CRPS	**0.6015**	0.6938	0.799	0.6995	0.6181	0.6318	0.6007	0.6018
(sd)	(0.00685)	(0.00687)	(0.0141)	(0.00971)	(0.00674)	(0.00778)	(0.00662)	(0.00691)
[p-val]		[7.27e-120]	[3.22e-115]	[1.02e-106]	[2.59e-86]	[2.75e-86]	[1.02e-37]	[1.23e-09]
RMSE_ext	**1.426**	1.549	1.446	1.429	1.447	1.466	1.426	1.426
(sd)	(0.0266)	(0.0273)	(0.0336)	(0.0266)	(0.0261)	(0.0264)	(0.0267)	(0.0266)
[p-val]		[1.11e-112]	[1.74e-15]	[2.86e-17]	[5.62e-70]	[6.58e-80]	[0.047]	[4.89e-08]
CRPS_ext	**0.8049**	0.8723	0.817	0.8137	0.8166	0.8269	0.8047	0.8052
(sd)	(0.0154)	(0.0153)	(0.02)	(0.0166)	(0.0148)	(0.0151)	(0.015)	(0.0153)
[p-val]		[1.37e-112]	[2.81e-15]	[3.33e-64]	[6.36e-66]	[6.8e-80]	[1.01e-05]	[1.39e-07]
RMSE_sum	**11.32**	11.81	14.69	16.23	18.64	18.53	11.32	11.32
(sd)	(1.4)	(1.36)	(1.58)	(1.6)	(1.8)	(1.79)	(1.41)	(1.4)
[p-val]		[8.43e-23]	[1.34e-56]	[1.99e-64]	[3.05e-68]	[5.04e-68]	[0.653]	[0.492]
CRPS_sum	**5.955**	6.323	8.643	9.929	11.35	11.3	5.953	5.953
(sd)	(0.65)	(0.653)	(1.06)	(1.18)	(1.34)	(1.34)	(0.642)	(0.652)
[p-val]		[2.73e-28]	[4.26e-61]	[5.57e-67]	[3.61e-69]	[8.45e-69]	[0.274]	[0.528]
RMSE_F	**0.2057**	0.6435		0.226	0.3205	0.3943	0.2058	0.2084
(sd)	(0.0123)	(0.0125)		(0.0162)	(0.00993)	(0.0182)	(0.0122)	(0.0124)
[p-val]		[2.84e-145]		[4.48e-34]	[4.54e-108]	[7.5e-103]	[0.381]	[8.77e-21]
RMSE_b	**0.311**	0.3542		0.6985	0.3203	0.3242	0.3111	0.3111
(sd)	(0.0113)	(0.0124)		(0.023)	(0.0111)	(0.0109)	(0.0111)	(0.0113)
[p-val]		[4.26e-76]		[1.11e-121]	[2.1e-31]	[1.07e-52]	[0.68]	[0.112]
sigma2	0.1145	0.4057	0.8496	0.3639	0.09053	0.0638	0.05827	0.1135
sigma2_b	0.07432	0.08767			0.07502	0.07534	0.05539	0.07455
time (s)	0.8152	0.008289	0.027	8.917	333.3	0.9111	5.682	0.08239

Table 1: Results for the grouped random effects and the mean function F = 'friedman3'. For all accuracy metrics (RMSE, CRPS, quantile loss), averages over the simulatiom runs are reported. Corresponding standard deviations are in parentheses. P-values are calculated using paired t-tests comparing the GPBoost algorithm to the other approaches. For the (co-)variance parameters, RMSEs are reported. 'GBPOOS' refers to the GPBoostOOS algorithm and 'GPBFixLR' denotes the GPBoost algorithm with the learning rate held fixed at 0.1. 'QL' denotes the quantile loss. Results for the "extrapolation" data sets, see Section 4.1, are denoted by '_ext' and results for the predictions of sums are denoted by '_sum'. The smallest values (excluding 'GBPOOS' and 'GPBFixLR') are in boldface. An empty value indicates that the required predictions or estimates cannot be calculated.

effect model which has a higher RMSE. Noteworthy is the considerably lower RMSE of the GPBoostOOS algorithm compared to all other methods. For the error variance σ^2, we observe larger differences. In particular, the GPBoost algorithm has an RMSE that is smaller than the ones of a linear mixed effects model, least squares boosting, and mboost, but a larger RMSE than the MERF and REEMtree algorithms. As can be seen in Figure 2, all methods have biased estimates for this error variance parameter, and the GPBoost algorithm has a downward bias. As discussed in Section 3.3, this finding is in line with the recent observation that state-of-the-art machine learning methods can interpolate the training data while at the same time having a low generalization error [Wyner et al., 2017, Belkin et al., 2019]. When estimating the covariance parameters on out-of-sample data using the GPBoostOOS algorithm, there is no downward bias anymore, and the RMSE of the error variance is smaller than the one of the GPBoost algorithm and also all other approaches. Note that the MERF algorithm also estimates covariance parameters using out-of-sample data. Finally, we also report the wall-clock time in seconds needed for training the different models. Not surprisingly, both least squares gradient boosting and the linear model are faster than the GPBoost algorithm. Further, we observe a large difference between the GPBoost and the MERF algorithm, with the GPBoost algorithm running more than 400 times faster than the MERF algorithm. The mboost

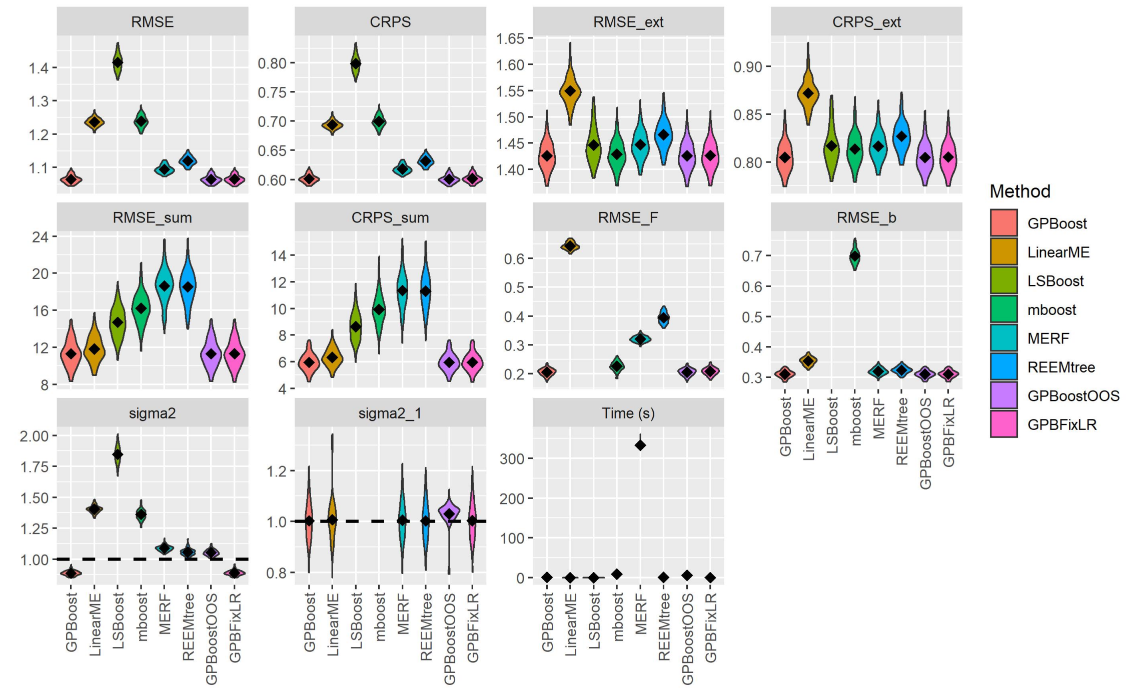

Figure 2: Violin plots illustrating the simulation results for the grouped random effects and the mean function F = 'friedman3'. The black rhombi represent means. For the (co-)variance parameters, the dashed lines represent the true values. If a plot is not shown, this means that the required predictions or estimates cannot be calculated. See the caption of Table 1 for more information.

algorithm runs approximately 10 times slower than GPBoost. Finally, when we fix the learning rate to 0.1 ('GPBFixLR'), the GPBoost algorithm is approximately 10 faster than the fully tuned version. Further, the predictive accuracy of the GPBoost with a learning rate of $\nu = 0.1$ is essentially equivalent to the one of the fully tuned GPBoost algorithm. I.e., if one is willing to sacrifice a tiny amount of predictive accuracy, one obtains an algorithm that runs 10 faster.

Considering the results for the 'hajjem' mean function reported in Table 5 in the appendix, we find qualitatively very similar results as for the 'friedman3' mean function. Further, as expected, the linear mixed effects model performs best in the case where the mean function is linear; see Table 6 in the appendix. Interestingly, the differences between the linear model and the GPBoost algorithm are of small, albeit significant, magnitude. Except for the linear model, the GPBoost algorithm significantly outperforms all other approaches.

We next discuss the results for the Gaussian process model reported in Table 2 and Figure 3. Apart from a linear Gaussian process model, we also consider two boosting approaches that model the spatial effect using deterministic base learners (LSBoost and mboost). We find that the GPBoost algorithm has significantly higher predictive accuracy compared to all alternative approaches in all metrics. We note that mboost results in very large errors for several simulation runs when doing spatial extrapolation. This is reflected in the very large average values, and it inflates variance estimates and thus p-values when doing paired t-tests. Further, the results for the 'hajjem' mean function reported in Table 7 in the appendix are very similar. The GPBoost algorithm significantly outperforms all other approaches in all predictive accuracy metrics. As expected, a linear model has the highest predictive accuracy when the data generating process is linear; see Table 8 in the appendix. Despite the relatively small sample size of $n = 500$, the differences in predictive accuracy between the linear Gaussian process model and the GPBoost algorithm are relatively small but mostly significant. Overall, in contrast to the larger sample size of the grouped random effects,

	GPBoost	LinearGP	LSBoost	mboost	GPBoostOOS	GPBFixLR
RMSE	**1.264**	1.362	1.375	1.543	1.266	1.265
(sd)	(0.0439)	(0.051)	(0.0496)	(0.0722)	(0.0421)	(0.0433)
[p-val]		[1.23e-48]	[5.53e-49]	[7.38e-67]	[0.26]	[0.162]
CRPS	**0.724**	0.7646	0.794	0.868	0.7136	0.724
(sd)	(0.0282)	(0.0279)	(0.0321)	(0.0401)	(0.0233)	(0.0276)
[p-val]		[1.56e-34]	[4.32e-49]	[5.47e-63]	[3.08e-19]	[0.886]
RMSE_ext	**1.43**	1.512	1.524	319500	1.431	1.43
(sd)	(0.106)	(0.102)	(0.148)	(1090000)	(0.106)	(0.105)
[p-val]		[1.11e-34]	[1.04e-15]	[0.00429]	[0.18]	[0.0722]
CRPS_ext	**0.8145**	0.8526	0.897	319500	0.8094	0.8148
(sd)	(0.0664)	(0.0587)	(0.107)	(1090000)	(0.0609)	(0.0661)
[p-val]		[1.12e-24]	[2.3e-19]	[0.00429]	[2.58e-06]	[0.196]
RMSE_sum	**11.59**	11.88	13.75	4518000	11.64	11.59
(sd)	(2.08)	(1.96)	(2.81)	(15500000)	(2.09)	(2.07)
[p-val]		[9.37e-05]	[7.6e-19]	[0.00429]	[0.0735]	[0.355]
CRPS_sum	**6.357**	6.488	8.26	3195000	6.333	6.359
(sd)	(1.06)	(0.979)	(1.91)	(10900000)	(1.02)	(1.06)
[p-val]		[0.00185]	[1.13e-24]	[0.00429]	[0.182]	[0.654]
RMSE_F	**0.4822**	0.6873			0.4827	0.4835
(sd)	(0.0661)	(0.0592)			(0.0642)	(0.0657)
[p-val]		[4.46e-58]			[0.532]	[0.0807]
RMSE_b	**0.6671**	0.6799			0.6683	0.6671
(sd)	(0.0448)	(0.0564)			(0.0459)	(0.045)
[p-val]		[0.000624]			[0.354]	[0.967]
sigma2	0.4282	0.4204	0.09895	1.215	0.2617	0.4188
sigma2_b	0.242	0.3107			0.2802	0.2426
rho	0.02693	0.03885			0.03823	0.02617
time (s)	5.102	0.5254	0.0318	0.863	10.24	3.927

Table 2: Results for the Gaussian process and the mean function F = 'friedman3'. For all accuracy metrics (RMSE, CRPS, quantile loss), averages over the simulatiom runs are reported. Corresponding standard deviations are in parentheses. P-values are calculated using paired t-tests comparing the GPBoost algorithm to the other approaches. For the (co-)variance parameters, RMSEs are reported. 'GBPOOS' refers to the GPBoostOOS algorithm and 'GPBFixLR' denotes the GPBoost algorithm with the learning rate held fixed at 0.1. 'QL' denotes the quantile loss. Results for the "extrapolation" data sets, see Section 4.1, are denoted by '_ext' and results for the predictions of sums are denoted by '_sum'. The smallest values (excluding 'GBPOOS' and 'GPBFixLR') are in boldface. An empty value indicates that the required predictions or estimates cannot be calculated.

the differences between the GPBoost and the GPBosstOOS algorithms in the accuracy of the error variance parameter σ^2 estimate are less pronounced. This is to be expected as the small sample size translates into high variance for the mean function estimate and, consequently, the out-of-sample parameter estimates for θ are also less efficient.

5 Real-world applications

In the following, we apply the GPBoost algorithm to two real-world data sets and compare its predictive accuracy to existing alternative methods. We consider data sets for both grouped random effects and a Gaussian process model.

5.1 Grouped random effects: wages data

For grouped random effects, we consider panel data from the National Longitudinal Survey of Young Working Women. The data consists of $28'534$ observations for $4'711$ young working women and was collected within the "National Longitudinal Survey" over the years 1968-1988. It can be

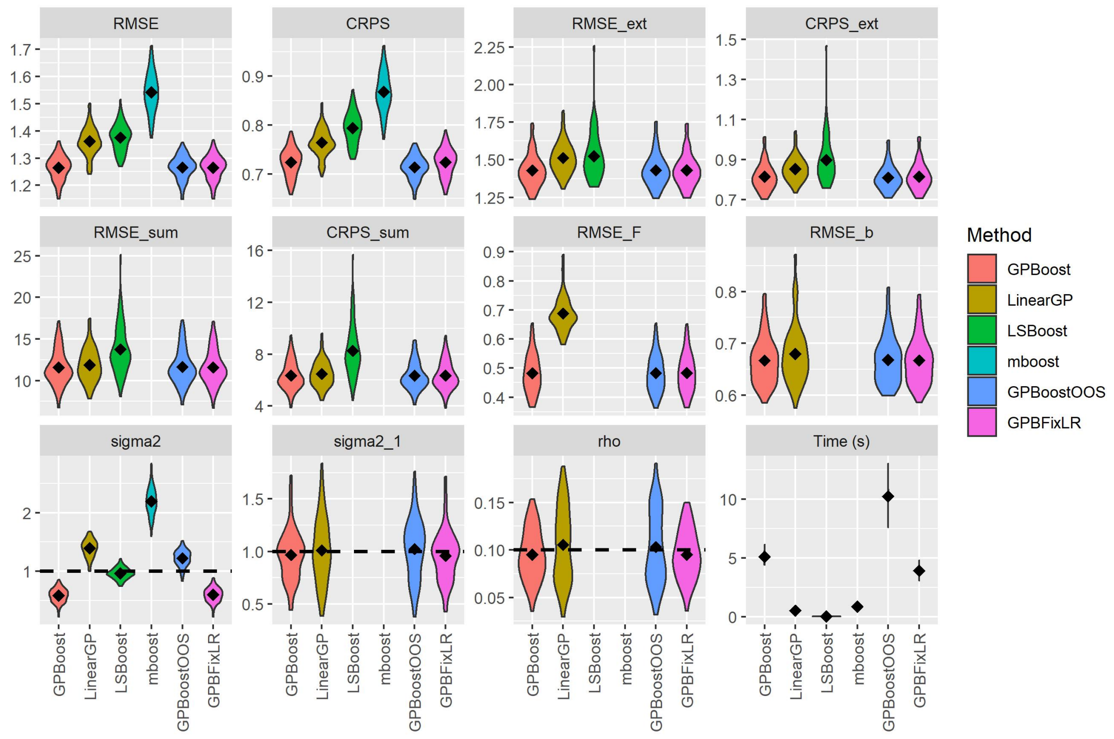

Figure 3: Violin plots illustrating the the simulation results for the Gaussian process and the mean function F = 'friedman3'. The black rhombi represent means. For the (co-)variance parameters, the dashed lines represent the true values. If a plot is not shown, this means that the corresponding estimates cannot be calculated. In addition, the '_ext'- and '_sum'-plots for 'mboost' are not shown since they contain very large values. See the caption of Table 2 for more information.

downloaded from `https://www.stata-press.com/data/r10/nlswork.dta`. The response variable is the logarithmic real wage, the persons ID number constitutes the grouping variable, and the data includes the following predictor variables: `age`, `ttl_exp` (total work experience), `tenure` (job tenure in years), `not_smsa` (1 if not SMSA), `south` (1 if south), `year` (interview year), `msp` (1 if married, spouse present), `nev_mar` (1 if never married), `collgrad` (1 if college graduate), `c_city` (1 if central city), `hours` (usual hours worked), `grade` (current grade completed), `ind_code` (industry of employment), `occ_code` (occupation), and `race` (1=white, 2=black, 3=other). The categorical variables `ind_code`, `occ_code`, `race`, and `year` are dummy coded. Further, we also include the square of `age`, `ttl_exp`, and `tenure` for the linear model.

We compare the predictive accuracy of the different approaches using nested 4-fold cross-validation. Specifically, all observations are partitioned into four disjoint sets, and in every fold, three of the sets are used as training and the other set as test data. Note that the test data sets contain both groups that are observed and unobserved in the training data. We compare the GPBoost algorithm to the same alternative methods as in the simulation study in Section 4.2: a linear mixed effects model, gradient boosting with a square loss including the grouping variable as a categorical variable (LSBoost), gradient boosting with ridge regression base learner for the grouped effects using the `mboost` R package, the RE-EM tree algorithm, and the MERF algorithm. Tuning parameters are chosen by doing an additional inner 4-fold cross-validation on every of the four training data sets and by minimizing the RMSE. We consider the same set of tuning parameters as in the simulated experiments; see Section 4.2.

The results are summarized in Table 3 and Figure 4. Table 3 reports the results in aggregate

	GPBoost	LinearME	LSBoost	mboost	MERF	REEMtree
RMSE	**0.2957**	0.3053	0.3131	0.3308	0.299	0.3221
CRPS	**0.151**	0.1586	0.1638	0.1734	0.1533	0.1672

Table 3: Comparison of predictive accuracy for the wages data. The smallest values are in boldface.

form over the different test sets, and Figure 4 graphically displays the results per fold. We find that the GPBoost algorithm has the highest predictive accuracy in terms of both the RMSE and the CRPS. As Figure 4 shows, this holds true for every fold. In particular, the GPBoost algorithm has higher predictive accuracy than both gradient boosting with a square loss when including the grouping variable as a categorical variable as well as a linear mixed effects model. The second best method is the MERF algorithm.

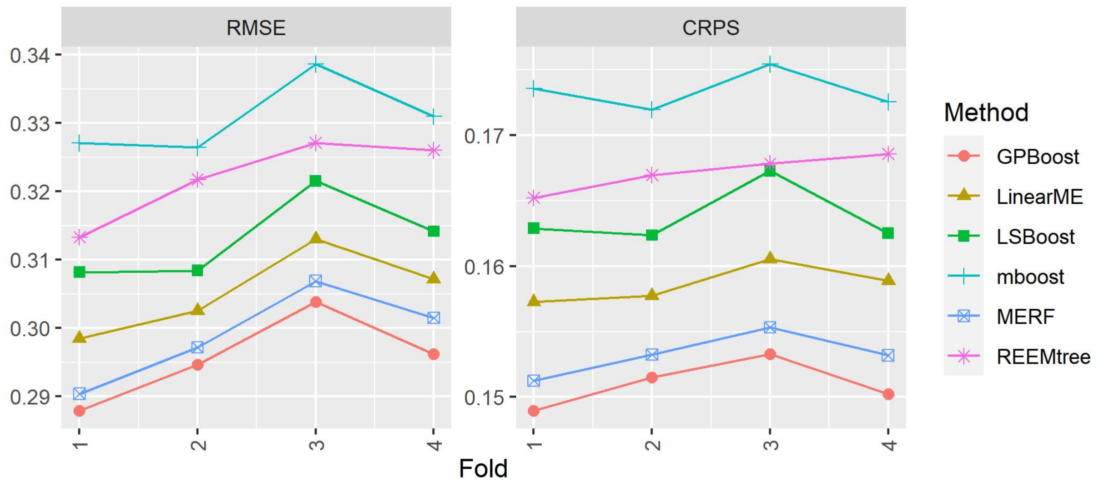

Figure 4: Comparison of predictive accuracy for the wages data.

5.2 Gaussian process model: house price data

We consider house price data for $25'357$ single family homes sold in Lucas County, Ohio. The data has originally been provided by the Spatial Econometrics Toolbox for Matlab, it is available in the `spData` R package [Bivand et al., 2008], and it has been previously studied by LeSage and Pace [2004], Bivand [2011], Dubé and Legros [2013]. The response variable is the logarithmic selling price. Further, the data includes the following predictor variables: `age`, `stories` (factor with levels {one, bilevel, multilvl, one+half, two, two+half, three}), `TLA` (total living area), `wall` (factor with levels {stucdrvt, ccbtile, metlvnyl, brick, stone, wood partbrk}), `beds` (number of bedrooms), `baths` (number of full baths), `halfbaths` (number of halfhbaths), `frontage` (lot frontage), `depth`, `garage` (factor with levels {no garage, basement, attached, detached, carport}), `garagesqft`, `rooms` (number of rooms), `lotsize`, `sdate` (year in which the house was sold, $1993 \leq$ `sdate` ≤ 1998), as well as longitude-latitude coordinates for the location. For the Gaussian process model with a linear mean function, we follow Bivand et al. [2008] and also include the square and cube of age as predictor variables, and as in Dubé and Legros [2013], we logarithmize the total living area and the lot size. The left plot of Figure 5 shows the observation locations and observed logarithmic prices. Further, the right plot of Figure 5 shows smoothed differences of log-prices from the global mean. The latter are obtained by, first, estimating a zero-mean Gaussian process to the differences of the log-prices from the global mean and, then, making spatial predictions using the posterior mean. We use a stationary exponential covariance function and a Vecchia approximation as documented below.

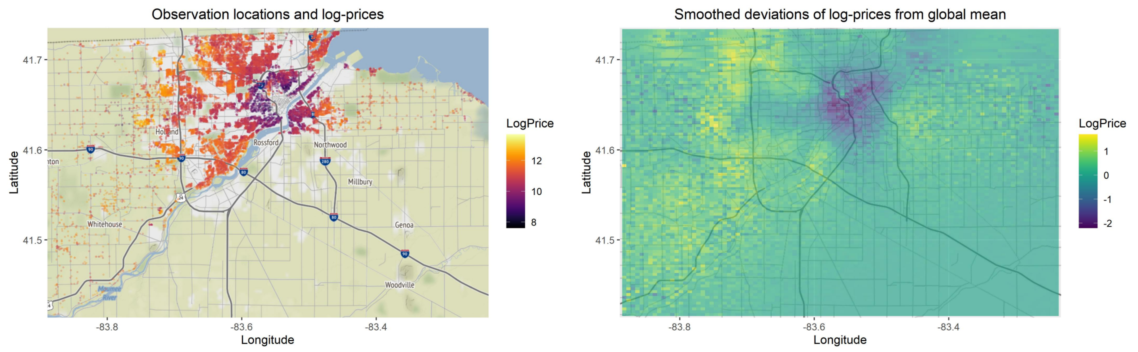

Figure 5: Illustration of house price data: map with observation locations and log-prices (left plot) and smoothed differences of log-prices from the global mean (right plot).

We measure the predictive accuracy by partitioning the data into expanding window training data sets and temporal out-of-sample test data sets. Specifically, learning is done on an expanding window containing all data up to the year $t-1$ and predictions are calculated for the next year t. We use the three years $t \in \{1996, 1997, 1998\}$ as test data sets for comparing the methods. Further, for every t, tuning parameters for the methods described below are chosen by additionally splitting the training data sets into two parts: inner training data containing all data up to year $t-2$ and validation data for the year $t-1$. Tuning parameters are then chosen by learning on the inner training data and minimizing the RMSE of predictions for the validation data sets. We consider the same grid of tuning parameters as in the simulated experiments in Section 4.2.

As in the simulated experiments in Section 4.1, we measure the accuracy of both points predictions and probabilistic predictions using the RMSE and the CRPS. In addition to univariate predictions, we also generate predictions for the total value of multiple objects as explained in the following. For every test set, we randomly select 100 times disjoints sets of 20 observations which are close together in space. Specifically, we randomly select a location and then determine its 19 nearest neighbors to obtain a set of 20 observations and then iteratively continue in the same manner with the remaining locations. For these sets of 20 objects, we then calculate predictive distributions for their sums as described in Section 4.1.[6] Further, we also evaluate the accuracy of $\alpha-$quantile predictions for $\alpha = 0.05$. This is motivated by the fact that, in practice, predictions for lower quantiles are required for risk management purposes such as, e.g., calculating the value-at-risk. Such quantile predictions are obtained by assuming Gaussian predictive distributions with the means and variances determined by the fixed and random effects. If a method does not contain random effects, we use the error variance. Quantile predictions are evaluated using the proper scoring rule [Gneiting et al., 2007]

$$S(y_s, \hat{y}) = (y_s - \hat{y})(\alpha - 1_{\{y_s \leq \hat{y}\}}), \tag{27}$$

where $\hat{y}$ denotes the predicted quantile and y_s the observed data.

We consider the same alternative approaches as in the simulation study: a linear Gaussian process, gradient boosting with a square loss, and gradient boosting with spline base learner for spatial effects using the `mboost` R package. For gradient boosting with a square loss and also the GPBoost algorithm, we include the coordinates as predictor variables in the mean function F. For comparison, we also report the results when excluding the coordinates from the mean function. For

[6]For simplicity, we predict the sum on the logarithmic scale. If the sum should be predicted on the original scale, simulation is required as the sum of dependent log-normal variables does not follow a standard distribution.

the linear Gaussian process model and the GPBoost algorithm, we use a stationary exponential covariance function as in (25).

Due to the relatively large sample size, we use the Vecchia approximation as outlined in Sections 3.4 and 3.5.1 for all Gaussian process-based models. Specifically, for training, we use the Vecchia approximation for the response variable with 50 nearest neighbors and a random ordering of the observations; see Section 3.4.1. For prediction, we use the result in Proposition 3.3 with the observed data ordered first, conditioning on observed data only when calculating the Vecchia approximation, and using 500 nearest neighbors. Further, for generating multivariate predictive distributions of dimension 20 for the predictions of sums, we use the same Vecchia approximation, but we condition on all data and not just the observed data. The latter is computationally more expensive, but it allows for obtaining more accurate, non-diagonal predictive covariance matrices; see Section 3.5.1 for more information.

	GPBoost	LinearGP	LSBoost	mboost	GPBoost_excl_coord	LSBoost_excl_coord
RMSE	**0.2657**	0.3554	0.2662	0.3329	0.2911	0.3378
CRPS	**0.1429**	0.1762	0.1432	0.174	0.1552	0.1794
QL	**0.03527**	0.0521	0.036	0.04686	0.03762	0.04607
RMSE_sum	**1.733**	2.52	1.945	3.138	2.094	3.736
CRPS_sum	**1.07**	1.341	1.235	1.848	1.262	2.304
QL_sum	**0.1899**	0.4453	0.2341	0.5173	0.2465	0.7048

Table 4: Comparison of predictive accuracy for the housing data. The smallest values are in boldface. Results for the predictions of sums are denoted by '_sum'. 'QL' denotes the quantile loss.

The results are reported in Table 4 and Figure 6. Table 4 reports the results in aggregate form over the different test sets, and Figure 6 graphically displays the results per test set. We observe that the GPBoost has the highest predictive accuracy for all metrics and across all methods. In particular, Table 4 shows that the GPBoost algorithm clearly outperforms a linear Gaussian process model and also the mboost algorithm in all predictive accuracy metrics for both univariate and areal sum predictions. For univariate predictions, the GPBoost algorithm also has a higher predictive accuracy than least squares boosting with the coordinates included in the mean function for all metrics, but the differences are relatively small. However, the GPBoost algorithm has considerably higher predictive accuracy compared to least squares boosting for the prediction of sums of several objects in all metrics: the RMSE ('RMSE_sum'), the CPRS ('CRPS_sum'), and the quantile loss ('QL_sum').

Further, the comparison between Gaussian process boosting without interactions between locations and the other predictor variables ('GPBoost_excl_coord') and the corresponding independent gradient boosting version with a square loss ('LSBoost_excl_coord') shows that there is important residual spatial variation also conditional on the effect of the predictor variables. Interestingly, the GPBoost algorithms with the coordinates also included in the mean function ('GPBoost') performs clearly better compared to the GPBoost algorithm when the coordinates are not included in the mean function ('GPBoost_excl_coord'). This is an indication that there are interaction effects between the predictor variables and the spatial locations.

For illustration, we plot in Figure 7 predicted random effects means and variances obtained using the GPBoost algorithm applied on the entire data set and when not including the locations in the fixed effects function. We use the of tuning parameters obtained on the last validation data set. Comparing the predicted mean field to the one of Figure 5, we see that in some areas the spatial effect is substantially different when factoring out the effect of the other predictor variables. As expected, prediction variances are high in areas with few observations.

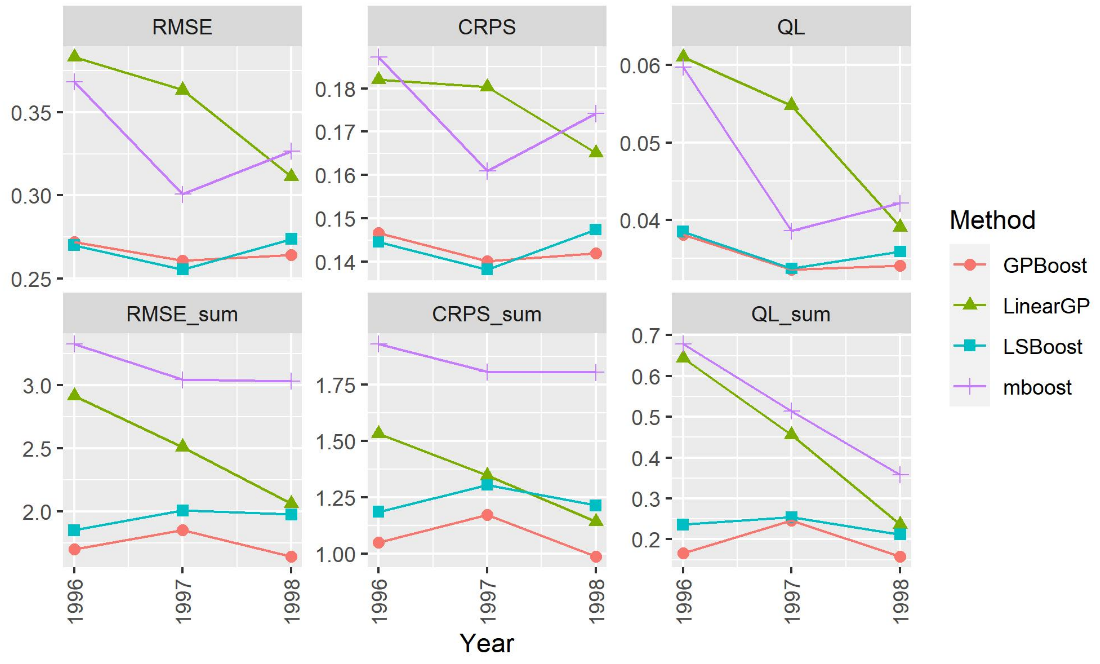

Figure 6: Comparison of predictive accuracy for the housing data.

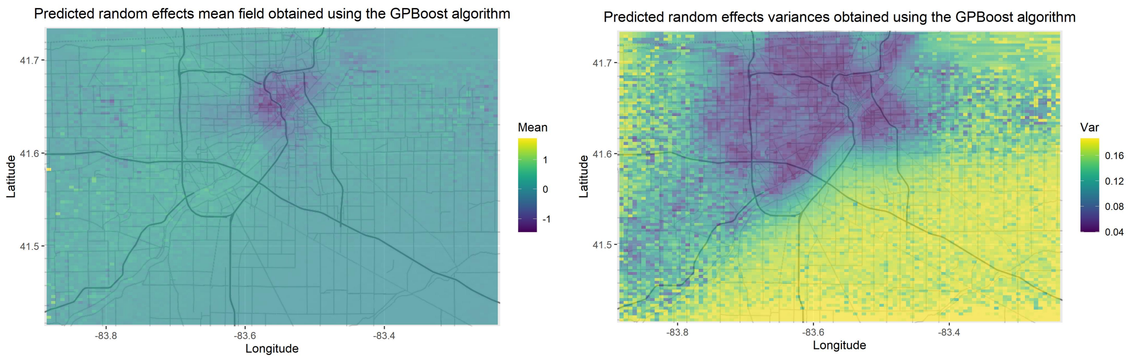
Predicted random effects mean field obtained using the GPBoost algorithm
Predicted random effects variances obtained using the GPBoost algorithm
Latitude
Longitude
Mean
Var

Figure 7: Predicted random effects means and variances obtained using the GPBoost algorithm when not including the locations in the fixed effects function.

6 Conclusion

We have introduced a novel way for combining boosting with Gaussian process and mixed effects models. This allows for relaxing, first, the linearity assumption for the mean function in Gaussian process and mixed effects models in a flexible non-parametric way and, second, the independence assumption made in most boosting algorithms. In simulation experiments and real-wold applications, we have shown that this leads to improved predictive accuracy compared to existing state-of-the-art methods. Future research should investigate how the GPBoost algorithm compares to other statistical and machine learning techniques, in particular linear models and independent least squares tree-boosting, on other data sets. Intuitively, we conjecture that the improvement in predictive accuracy of the GPBoost algorithm over least squares tree-boosting is the larger, the larger the effective sample size of the random effects is compared to the actual sample size, i.e. the more groups with large effects or the larger and faster decaying the covariance function of a Gaussian process. Further, we are currently investigating how our approach can be extended to non-Gaussian data using, e.g., Laplace approximations. Future research is also required for convergence results concerning the generalization error as wells as finite-sample and asymptotic properties of the estimators obtained from the GPBoost algorithm.

Acknowledgments

We thank Hansruedi Künsch for very helpful comments on a first version of this article. This research was partially supported by the Swiss Innovation Agency - Innosuisse (grant numbers '25746.1 PFES-ES' and '28408.1 PFES-ES').

References

S.-I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

S. Banerjee, B. P. Carlin, and A. E. Gelfand. *Hierarchical modeling and analysis for spatial data.* Chapman and Hall/CRC, 2014.

M. Belkin, S. Ma, and S. Mandal. To understand deep learning we need to understand kernel learning. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 541–549, 2018.

M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32): 15849–15854, 2019.

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

R. Bivand. After "raising the bar": applied maximum likelihood estimation of families of models in spatial econometrics. Technical report, Norwegian School of Economics, Department of Economics, 2011.

R. S. Bivand, E. J. Pebesma, V. Gomez-Rubio, and E. J. Pebesma. *Applied spatial data analysis with R*, volume 747248717. Springer, 2008.

L. Breiman. Arcing classifiers. *Annals of Statistics*, pages 801–824, 1998.

L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen. *Classification and regression trees.* CRC press, 1984.

P. Bühlmann and T. Hothorn. Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science*, pages 477–505, 2007.

P. Bühlmann and B. Yu. Boosting with the l 2 loss: regression and classification. *Journal of the American Statistical Association*, 98(462):324–339, 2003.

P. Bühlmann et al. Boosting for high-dimensional linear models. *The Annals of Statistics*, 34(2): 559–583, 2006.

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

N. Cressie and C. K. Wikle. *Statistics for spatio-temporal data*. John Wiley & Sons, 2015.

J. A. Dambon, F. Sigrist, and R. Furrer. Maximum likelihood estimation of spatially varying coefficient models for large data with an application to real estate price prediction. *Spatial Statistics*, 41:100470, 2021. ISSN 2211-6753. doi: https://doi.org/10.1016/j.spasta.2020.100470.

A. Datta, S. Banerjee, A. O. Finley, and A. E. Gelfand. Hierarchical nearest-neighbor gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, 111(514):800–812, 2016.

T. Davis. Csparse: a concise sparse matrix package, 2005. URL `http://www.cise.ufl.edu/research/sparse/CSparse`.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

J. Dubé and D. Legros. Dealing with spatial data pooled over time in statistical models. *Letters in Spatial and Resource Sciences*, 6(1):1–18, 2013.

J. Elith, J. R. Leathwick, and T. Hastie. A working guide to boosted regression trees. *Journal of Animal Ecology*, 77(4):802–813, 2008.

A. O. Finley, A. Datta, B. D. Cook, D. C. Morton, H. E. Andersen, and S. Banerjee. Efficient algorithms for bayesian nearest neighbor gaussian processes. *Journal of Computational and Graphical Statistics*, 28(2):401–414, 2019.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156. Bari, Italy, 1996.

J. Friedman, T. Hastie, R. Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2):337–407, 2000.

J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, pages 1–67, 1991.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

W. Fu and J. S. Simonoff. Unbiased regression trees for longitudinal and clustered data. *Computational Statistics & Data Analysis*, 88:53–74, 2015.

G.-A. Fuglstad, D. Simpson, F. Lindgren, and H. Rue. Does non-stationary spatial data always require non-stationary random fields? *Spatial Statistics*, 14:505–531, 2015.

A. E. Gelfand, H.-J. Kim, C. Sirmans, and S. Banerjee. Spatial modeling with spatially varying coefficient processes. *Journal of the American Statistical Association*, 98(462):387–396, 2003.

T. Gneiting, F. Balabdaoui, and A. E. Raftery. Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(2):243–268, 2007.

V. P. Godambe. An optimum property of regular maximum likelihood estimation. *The Annals of Mathematical Statistics*, 31(4):1208–1211, 1960.

A. Groll and G. Tutz. Regularization for generalized additive mixed models by likelihood-based boosting. *Methods of Information in Medicine*, 51(02):168–177, 2012.

G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

J. Guinness. Permutation and grouping methods for sharpening gaussian process approximations. *Technometrics*, 60(4):415–429, 2018.

J. Guinness. Gaussian process learning via fisher scoring of vecchia's approximation. *arXiv preprint arXiv:1905.08374*, 2019.

A. Hajjem, F. Bellavance, and D. Larocque. Mixed effects regression trees for clustered data. *Statistics & probability letters*, 81(4):451–459, 2011.

A. Hajjem, F. Bellavance, and D. Larocque. Mixed-effects random forest for clustered data. *Journal of Statistical Computation and Simulation*, 84(6):1313–1328, 2014.

T. Hastie and R. Tibshirani. Generalized additive models. *Statistical Science*, 1(3):297–310, 1986.

B. Hofner, A. Mayr, N. Robinzonov, and M. Schmid. Model-based boosting in R: A hands-on tutorial using the R package mboost. *Computational Statistics*, 29:3–35, 2014.

T. Hothorn, P. Bühlmann, T. Kneib, M. Schmid, and B. Hofner. Model-based boosting 2.0. *Journal of Machine Learning Research*, 11(Aug):2109–2113, 2010.

R. Johnson and T. Zhang. Learning nonlinear functions using regularized greedy forest. *IEEE transactions on pattern analysis and machine intelligence*, 36(5):942–954, 2013.

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

M. Katzfuss and J. Guinness. A general framework for vecchia approximations of gaussian processes. *arXiv preprint arXiv:1708.06302*, 2017.

M. Katzfuss, J. Guinness, W. Gong, and D. Zilber. Vecchia approximations of gaussian-process predictions. *arXiv preprint arXiv:1805.03309*, 2018.

G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3149–3157, 2017.

M. C. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

N. M. Laird, J. H. Ware, et al. Random-effects models for longitudinal data. *Biometrics*, 38(4): 963–974, 1982.

J. P. LeSage and R. K. Pace. Models for spatially dependent missing data. *The Journal of Real Estate Finance and Economics*, 29(2):233–254, 2004.

H. Lu, S. P. Karimireddy, N. Ponomareva, and V. Mirrokni. Accelerating gradient boosting machine. *arXiv preprint arXiv:1903.08708*, 2019.

L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting algorithms as gradient descent. In *Advances in neural information processing systems*, pages 512–518, 2000.

C. E. McCulloch and J. M. Neuhaus. Misspecifying the shape of a random effects distribution: why getting it wrong may not matter. *Statistical science*, pages 388–402, 2011.

Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2004.

D. Nielsen. Tree boosting with xgboost-why does xgboost win" every" machine learning competition? Master's thesis, NTNU, 2016.

A. Pande, L. Li, J. Rajeswaran, J. Ehrlinger, U. B. Kogalur, E. H. Blackstone, and H. Ishwaran. Boosted multivariate trees for longitudinal data. *Machine learning*, 106(2):277–305, 2017.

J. Pinheiro and D. Bates. *Mixed-effects models in S and S-PLUS.* Springer Science & Business Media, 2006.

L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6639–6649. Curran Associates, Inc., 2018.

H. Rue and L. Held. Discrete spatial variation. *Handbook of spatial statistics*, pages 171–200, 2010.

M. J. Saberian, H. Masnadi-Shirazi, and N. Vasconcelos. Taylorboost: First and second-order boosting algorithms with explicit margin control. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2929–2934. IEEE, 2011.

A. M. Schmidt and P. Guttorp. Flexible spatial covariance functions. *Spatial Statistics*, page 100416, 2020.

R. J. Sela and J. S. Simonoff. Re-em trees: a data mining approach for longitudinal and clustered data. *Machine learning*, 86(2):169–207, 2012.

R. H. Shumway and D. S. Stoffer. *Time series analysis and its applications: with R examples.* Springer, 2017.

F. Sigrist. KTBoost: Combined Kernel and Tree Boosting. *arXiv preprint arXiv:1902.03999*, 2019.

F. Sigrist. Gradient and newton boosting for classification and regression. *Expert Systems With Applications*, 2021. (in press).

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

M. L. Stein. *Interpolation of spatial data: some theory for kriging.* Springer Science & Business Media, 1999.

G. Tutz and F. Reithinger. A boosting approach to flexible semiparametric mixed models. *Statistics in medicine*, 26(14):2872–2900, 2007.

C. Varin, N. Reid, and D. Firth. An overview of composite likelihood methods. *Statistica Sinica*, pages 5–42, 2011.

A. V. Vecchia. Estimation and model identification for continuous spatial processes. *Journal of the Royal Statistical Society: Series B (Methodological)*, 50(2):297–312, 1988.

C. K. Williams and C. E. Rasmussen. *Gaussian processes for machine learning.* MIT Press Cambridge, MA, 2006.

S. N. Wood. *Generalized additive models: an introduction with R.* Chapman and Hall/CRC, 2017.

H. Wu and J.-T. Zhang. *Nonparametric regression methods for longitudinal data analysis: mixed-effects modeling approaches*, volume 515. John Wiley & Sons, 2006.

A. J. Wyner, M. Olson, J. Bleich, and D. Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *Journal of Machine Learning Research*, 18(48):1–33, 2017.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

Appendices

A Which optimization problems are solved by the MERT/MERF and RE-EM tree algorithms?

The MERT/MERF [Hajjem et al., 2011, 2014] and RE-EM tree [Sela and Simonoff, 2012, Fu and Simonoff, 2015] algorithms are heuristically motivated algorithms for combining trees and random forests with grouped random effects models. Unfortunately, it is nowhere in the literature explained which optimization problems these algorithms try to solve. In this section, we shed some light on this.

A.1 The EM algorithm for linear mixed effects models

Both approaches claim to be based on the EM algorithm [Dempster et al., 1977]. Using the notation of Section 2, the E-step in iteration t of an EM algorithm for mixed effect models works by, first, finding a maximizer for the mean function F of the multivariate normal likelihood given the current estimate for the covariance matrix Ψ_t:

$$\hat{F}_{t+1} = \underset{F}{\operatorname{argmin}}(y - F)^T {\Psi_t}^{-1}(y - F), \tag{28}$$

second, conditional on this obtaining predictions for the random effects

$$\hat{b}_{t+1} = Z\Sigma_t Z^T {\Psi_t}^{-1}(y - \hat{F}_{t+1}),$$

and then using these two quantities to calculate the expectation of the full data log-likelihood; see e.g. Laird et al. [1982] and Wu and Zhang [2006]. The EM algorithm then proceeds with the M-step by maximizing this expected full data log-likelihood to obtain an estimate for Ψ_{t+1} or its parameters θ_{t+1}.

A.2 MERT/MERF algorithms

In contrast, in the "E-step" of the MERT and MERF algorithms of Hajjem et al. [2011] and Hajjem et al. [2014], the mean function $\hat{F}_{t+1}$ is not obtained as maximizer of the multivariate normal likelihood given Ψ_t as in (28). Rather, they, first, use an independent normal likelihood obtained after subtracting predicted values of the random effects $\hat{b}_t$ of the previous iteration from the response variable y to learn the mean function using a regression tree or a random forest:

$$\hat{F}_{t+1} = \underset{F}{\operatorname{argmin}} \frac{1}{2}(y - F - \hat{b}_t)^T(y - F - \hat{b}_t),$$

and, second, obtain predictions for the random effects:

$$\hat{b}_{t+1} = Z\Sigma_t Z^T {\Psi_t}^{-1}(y - \hat{F}_{t+1}).$$

The M-step is then analogous to a correctly specified EM algorithm. It is thus unclear whether and to which quantities these algorithms converge as they do not correspond to correctly specified EM algorithms. In our experiments in Sections 4 and 5, we do not obtain convergence for the MERF algorithm of Hajjem et al. [2014].

A.3 RE-EM tree algorithm

The algorithm of Sela and Simonoff [2012] and Fu and Simonoff [2015] iterates between, first, estimating the structure of a tree using an independent normal likelihood obtained after subtracting predicted values of the random effects from the response and, second, jointly estimating the leaf values and covariance parameters using a classical linear mixed effects model. This is clearly not an EM algorithm as it does not involve an E-step that calculates an expectation of a full data log-likelihood. It can, however, be interpreted as a component-wise, or coordinate descent, minimization algorithm that iterates between finding an optimizer for (parts of) F, the covariance parameters θ, and the random effects b. This can be seen by first noting that, regarding F and θ, the minimization problem

$$(\hat{F},\hat{\theta}) = \underset{(F,\theta)}{\operatorname{argmin}} \frac{1}{2}(y-F)^T\Psi^{-1}(y-F) + \frac{1}{2}\log\det(\Psi) \tag{29}$$

is equivalent to

$$(\hat{F},\hat{\theta},\hat{b}) = \underset{(F,\theta,b)}{\operatorname{argmin}} \frac{1}{2\sigma^2}(y-F-Zb)^T(y-F-Zb) + \frac{1}{2}b^T\Sigma^{-1}b + \frac{1}{2}\log\det(\Psi)\,. \tag{30}$$

Further, the componentwise minimizer for b given F and θ corresponds to the best linear unbiased estimator for b:

$$\begin{aligned}\hat{b} =&(Z^TZ+\sigma^2\Sigma^{-1})^{-1}Z^T(y-F)\\ =&\Sigma Z^T(Z\Sigma Z^T+\sigma^2 I_n)^{-1}(y-F).\end{aligned}$$

Using the notation of Section 3.1, $F(\cdot) = h(\cdot;\alpha)^T\gamma$, where α denotes the splits of a tree and γ the leaf values, one iteration of the RE-EM tree algorithm can thus be written as

$$\begin{aligned}\hat{\alpha}_{t+1} =& \underset{\alpha}{\operatorname{argmin}}\, L\left(F = h(\cdot;\alpha)^T\hat{\gamma}_t, \hat{\theta}_t, \hat{b}_t\right),\\ \hat{\theta}_{t+1} =& \underset{\theta}{\operatorname{argmin}}\, L\left(F = h(\cdot;\hat{\alpha}_{t+1})^T\hat{\gamma}_t, \theta, \hat{b}_t\right),\\ (\hat{\gamma}_{t+1},\hat{b}_{t+1}) =& \underset{\gamma,b}{\operatorname{argmin}}\, L\left(F = h(\cdot;\hat{\alpha}_{t+1})^T\gamma, \hat{\theta}_{t+1}, b\right),\end{aligned}$$

where

$$L\,(F,\theta,b) = \frac{1}{2\sigma^2}(y-F-Zb)^T(y-F-Zb) + \frac{1}{2}b^T\Sigma^{-1}b + \frac{1}{2}\log\det(\Psi)\,.$$

I.e., this corresponds to one iteration of a coordinate descent algorithm applied to the minimization problem in (30) which is equivalent to (29).

B Proofs for the results involving the Vecchia approximation

Proof of Proposition 3.1. The negative log-likelihood of the Vecchia approximation in (16) is given by

$$\tilde{L}(y,F,\theta) = \frac{1}{2}(y-F)^T\tilde{\Psi}^{-1}(y-F) + \frac{1}{2}\sum_{i=1}^n \log D_i + \frac{n}{2}\log(2\pi). \tag{31}$$

It follows that

$$\frac{\partial\tilde{L}(y,F,\theta)}{\partial\theta_k} = \frac{1}{2\sigma^2}(y-F)^T\frac{\partial}{\partial\theta_k}\tilde{\Psi}^{-1}(y-F) + \frac{1}{2}\sum_{i=1}^n\frac{1}{D_i}\frac{\partial D_i}{\partial\theta_k}. \tag{32}$$

Further, we have

$$\frac{\partial}{\partial\theta_k}\tilde{\Psi}^{-1} = \frac{\partial B^T}{\partial\theta_k}D^{-1}B + B^TD^{-1}\frac{\partial B}{\partial\theta_k} - B^TD^{-1}\frac{\partial D}{\partial\theta_k}D^{-1}B.$$

We thus obtain the result in the proposition by noting that

$$(y-F)^T\frac{\partial}{\partial\theta_k}\tilde{\Psi}^{-1}(y-F) = 2u_k^Tu - u^T\frac{\partial D}{\partial\theta_k}u,$$

where u and u_k are given in (18). □

Proof of Proposition 3.2. We have

$$\frac{\partial \tilde{\Psi}}{\partial \theta_k} = -B^{-1}\frac{\partial B}{\partial \theta_k}B^{-1}DB^{-T} - B^{-1}DB^{-T}\frac{\partial B^T}{\partial \theta_k}B^{-T} + B^{-1}\frac{\partial D}{\partial \theta_k}B^{-T}.$$

It follows that

$$\tilde{\Psi}^{-1}\frac{\partial \tilde{\Psi}}{\partial \theta_k} = -B^T D^{-1}\frac{\partial B}{\partial \theta_k}B^{-1}DB^{-T} - \frac{\partial B^T}{\partial \theta_k}B^{-T} + B^T D^{-1}\frac{\partial D}{\partial \theta_k}B^{-T}.$$

We thus have

$$\begin{aligned}\tilde{\Psi}^{-1}\frac{\partial \tilde{\Psi}}{\partial \theta_k}\tilde{\Psi}^{-1}\frac{\partial \tilde{\Psi}}{\partial \theta_l} =& B^T D^{-1}\frac{\partial B}{\partial \theta_k}B^{-1}\frac{\partial B}{\partial \theta_l}B^{-1}DB^{-T} + \frac{\partial B^T}{\partial \theta_k}D^{-1}\frac{\partial B}{\partial \theta_l}B^{-1}DB^{-T}\\ &- B^T D^{-1}\frac{\partial D}{\partial \theta_k}D^{-1}\frac{\partial B}{\partial \theta_l}B^{-1}DB^{-T}\\ &+ B^T D^{-1}\frac{\partial B}{\partial \theta_k}B^{-1}DB^{-T}\frac{\partial B^T}{\partial \theta_l}B^{-T} + \frac{\partial B^T}{\partial \theta_k}B^{-T}\frac{\partial B^T}{\partial \theta_l}B^{-T}\\ &- B^T D^{-1}\frac{\partial D}{\partial \theta_k}B^{-T}\frac{\partial B^T}{\partial \theta_l}B^{-T}\\ &- B^T D^{-1}\frac{\partial B}{\partial \theta_k}B^{-1}\frac{\partial D}{\partial \theta_l}B^{-T} - \frac{\partial B^T}{\partial \theta_k}D^{-1}\frac{\partial D}{\partial \theta_l}B^{-T}\\ &+ B^T D^{-1}\frac{\partial D}{\partial \theta_k}D^{-1}\frac{\partial D}{\partial \theta_l}B^{-T}.\end{aligned}$$

Due to the cyclicality of the trace, it follows that

$$\begin{aligned}\operatorname{tr}\left(\tilde{\Psi}^{-1}\frac{\partial \tilde{\Psi}}{\partial \theta_k}\tilde{\Psi}^{-1}\frac{\partial \tilde{\Psi}}{\partial \theta_l}\right) =& \operatorname{tr}\left(\frac{\partial B}{\partial \theta_k}B^{-1}\frac{\partial B}{\partial \theta_l}B^{-1}\right) + \operatorname{tr}\left(\frac{\partial B^T}{\partial \theta_k}D^{-1}\frac{\partial B}{\partial \theta_l}B^{-1}DB^{-T}\right)\\ &- \operatorname{tr}\left(\frac{\partial D}{\partial \theta_k}D^{-1}\frac{\partial B}{\partial \theta_l}B^{-1}\right)\\ &+ \operatorname{tr}\left(D^{-1}\frac{\partial B}{\partial \theta_k}B^{-1}DB^{-T}\frac{\partial B^T}{\partial \theta_l}\right) + \operatorname{tr}\left(\frac{\partial B^T}{\partial \theta_k}B^{-T}\frac{\partial B^T}{\partial \theta_l}B^{-T}\right)\\ &- \operatorname{tr}\left(D^{-1}\frac{\partial D}{\partial \theta_k}B^{-T}\frac{\partial B^T}{\partial \theta_l}\right)\\ &- \operatorname{tr}\left(D^{-1}\frac{\partial B}{\partial \theta_k}B^{-1}\frac{\partial D}{\partial \theta_l}\right) - \operatorname{tr}\left(\frac{\partial B^T}{\partial \theta_k}D^{-1}\frac{\partial D}{\partial \theta_l}B^{-T}\right)\\ &+ \operatorname{tr}\left(D^{-1}\frac{\partial D}{\partial \theta_k}D^{-1}\frac{\partial D}{\partial \theta_l}\right).\end{aligned}$$

Since $\frac{\partial B}{\partial \theta_k}$ is lower triangular with zeros on the diagonal, we obtain

$$\begin{aligned}\operatorname{tr}\left(\tilde{\Psi}^{-1}\frac{\partial \tilde{\Psi}}{\partial \theta_k}\tilde{\Psi}^{-1}\frac{\partial \tilde{\Psi}}{\partial \theta_l}\right) =& \operatorname{tr}\left(\frac{\partial B^T}{\partial \theta_k}D^{-1}\frac{\partial B}{\partial \theta_l}B^{-1}DB^{-T}\right) + \operatorname{tr}\left(D^{-1}\frac{\partial B}{\partial \theta_k}B^{-1}DB^{-T}\frac{\partial B^T}{\partial \theta_l}\right)\\ &+ \operatorname{tr}\left(D^{-1}\frac{\partial D}{\partial \theta_k}D^{-1}\frac{\partial D}{\partial \theta_l}\right)\\ =& 2\operatorname{tr}\left(B^{-T}\frac{\partial B^T}{\partial \theta_k}D^{-1}\frac{\partial B}{\partial \theta_l}B^{-1}D\right) + \operatorname{tr}\left(D^{-1}\frac{\partial D}{\partial \theta_k}D^{-1}\frac{\partial D}{\partial \theta_l}\right),\end{aligned}$$

and the statement in (19) follows. □

Proof of Proposition 3.3. We have

$$\begin{pmatrix} B & 0 \\ B_{po} & B_p \end{pmatrix}^T \begin{pmatrix} D^{-1} & 0 \\ 0 & {D_p}^{-1} \end{pmatrix} \begin{pmatrix} B & 0 \\ B_{po} & B_p \end{pmatrix} = \begin{pmatrix} B_o^T {D_o}^{-1} B_o + B_{po}^T {D_p}^{-1} B_{po} & B_{po}^T {D_p}^{-1} B_p \\ B_p^T {D_p}^{-1} B_{po} & B_p^T {D_p}^{-1} B_p \end{pmatrix}.$$

Since

$$\left(B_p^T {D_p}^{-1} B_p\right)^{-1} B_p^T {D_p}^{-1} B_{po} = B_p^{-1} B_{po},$$

the result follows from Theorem 12.2 in Rue and Held [2010]. □

Proof of Proposition 3.4. We have

$$\begin{pmatrix} B_p & 0 \\ B_{op} & B_o \end{pmatrix}^T \begin{pmatrix} {D_p}^{-1} & 0 \\ 0 & {D_o}^{-1} \end{pmatrix} \begin{pmatrix} B_p & 0 \\ B_{op} & B_o \end{pmatrix} = \begin{pmatrix} B_p^T {D_p}^{-1} B_p + B_{op}^T {D_o}^{-1} B_{op} & B_{op}^T {D_o}^{-1} B_o \\ B_o^T {D_o}^{-1} B_{op} & B_o^T {D_o}^{-1} B_o \end{pmatrix},$$

and the result follows from Theorem 12.2 in Rue and Held [2010]. □

C Additional results for the simulated experiments: other mean functions

In the following, we present the results for the simulated experiments in Section 4 for the two mean functions which are not reported in the body of the article.

	GPBoost	LinearME	LSBoost	mboost	MERF	REEMtree	GPBoostOOS	GPBFixLR
RMSE	**1.1**	1.342	1.447	1.334	1.107	1.17	1.1	1.102
(sd)	(0.0145)	(0.0181)	(0.0284)	(0.0215)	(0.0138)	(0.0155)	(0.0146)	(0.0142)
[p-val]		[2.26e-129]	[2.33e-112]	[1.2e-111]	[8.01e-19]	[7.22e-84]	[0.475]	[1.98e-05]
CRPS	**0.621**	0.7475	0.8144	0.749	0.6231	0.6579	0.6184	0.6213
(sd)	(0.00761)	(0.00911)	(0.016)	(0.0117)	(0.00724)	(0.00822)	(0.00713)	(0.00738)
[p-val]		[1.04e-127]	[6.77e-111]	[1.27e-109]	[1.79e-09]	[4.11e-83]	[3.34e-33]	[0.0769]
RMSE_ext	**1.457**	1.635	1.496	1.519	1.462	1.505	1.457	1.459
(sd)	(0.0293)	(0.0282)	(0.0452)	(0.0285)	(0.0291)	(0.0283)	(0.0293)	(0.0293)
[p-val]		[3.14e-121]	[5.79e-22]	[2.55e-86]	[7.6e-14]	[7.44e-80]	[0.859]	[5.18e-07]
CRPS_ext	**0.8223**	0.9165	0.8436	0.8605	0.8243	0.8479	0.8214	0.8228
(sd)	(0.0169)	(0.0151)	(0.027)	(0.0171)	(0.0163)	(0.016)	(0.0161)	(0.0168)
[p-val]		[5.19e-119]	[9.36e-20]	[1.36e-91]	[9.2e-10]	[9.32e-78]	[1.85e-10]	[0.000286]
RMSE_sum	**11.42**	12.02	14.85	16.02	18.46	18.44	11.4	11.42
(sd)	(1.27)	(1.22)	(1.59)	(1.56)	(2.08)	(2.1)	(1.26)	(1.27)
[p-val]		[2.1e-24]	[2.83e-48]	[3.81e-58]	[2.35e-60]	[2.31e-60]	[6e-04]	[0.754]
CRPS_sum	**6.064**	6.517	8.792	9.695	11.21	11.19	6.046	6.064
(sd)	(0.639)	(0.621)	(1.09)	(1.12)	(1.43)	(1.45)	(0.623)	(0.637)
[p-val]		[3.09e-31]	[1.18e-54]	[4.44e-63]	[4.98e-63]	[2.67e-62]	[0.000244]	[0.864]
RMSE_F	**0.336**	0.8141		0.5469	0.3579	0.5093	0.3366	0.3399
(sd)	(0.0238)	(0.0197)		(0.0241)	(0.0209)	(0.0221)	(0.0241)	(0.0244)
[p-val]		[1.95e-141]		[1.85e-101]	[1.62e-24]	[3.42e-91]	[0.341]	[5.84e-06]
RMSE_b	**0.3194**	0.3793		0.6936	0.3262	0.3363	0.3193	0.3197
(sd)	(0.0109)	(0.0136)		(0.0221)	(0.0143)	(0.0118)	(0.0112)	(0.0109)
[p-val]		[9.35e-82]		[8.2e-121]	[2.51e-11]	[6.95e-47]	[0.359]	[0.0923]
sigma2	0.1737	0.6574	0.8399	0.6231	0.1207	0.1457	0.1399	0.1515
sigma2_b	0.07447	0.07963			0.07462	0.07507	0.1081	0.07556
time (s)	0.3243	0.01545	0.08928	14.12	415.3	1.459	2.116	0.1292

Table 5: Results for the grouped random effects and the mean function F = 'hajjem'. For all accuracy metrics (RMSE, CRPS, quantile loss), averages over the simulatiom runs are reported. Corresponding standard deviations are in parentheses. P-values are calculated using paired t-tests comparing the GPBoost algorithm to the other approaches. For the (co-)variance parameters, RMSEs are reported. 'GBPOOS' refers to the GPBoostOOS algorithm and 'GPBFixLR' denotes the GPBoost algorithm with the learning rate held fixed at 0.1. 'QL' denotes the quantile loss. Results for the "extrapolation" data sets, see Section 4.1, are denoted by '_ext' and results for the predictions of sums are denoted by '_sum'. The smallest values (excluding 'GBPOOS' and 'GPBFixLR') are in boldface. An empty value indicates that the required predictions or estimates cannot be calculated.

	GPBoost	LinearME	LSBoost	mboost	MERF	REEMtree	GPBoostOOS	GPBFixLR
RMSE	1.049	**1.044**	1.398	1.225	1.139	1.087	1.049	1.059
(sd)	(0.0107)	(0.0105)	(0.0252)	(0.0177)	(0.0128)	(0.0126)	(0.0107)	(0.0112)
[p-val]		[1.18e-49]	[1.52e-120]	[5.89e-112]	[1.14e-115]	[1.06e-82]	[0.806]	[1.47e-66]
CRPS	0.5917	**0.589**	0.789	0.6916	0.6429	0.6142	0.5917	0.5978
(sd)	(0.0061)	(0.00597)	(0.0144)	(0.0102)	(0.00724)	(0.0073)	(0.00605)	(0.0065)
[p-val]		[1.54e-48]	[4.06e-120]	[1.6e-111]	[3.27e-115]	[1.08e-83]	[0.0791]	[1.82e-68]
RMSE_ext	1.418	**1.415**	1.448	1.424	1.481	1.445	1.418	1.425
(sd)	(0.0256)	(0.0256)	(0.0356)	(0.0259)	(0.0255)	(0.0257)	(0.0256)	(0.0256)
[p-val]		[5.48e-40]	[4.61e-19]	[8.9e-44]	[5.05e-104]	[1.07e-78]	[0.736]	[4.59e-52]
CRPS_ext	0.8005	**0.7986**	0.8183	0.8106	0.8359	0.8153	0.8004	0.8043
(sd)	(0.0143)	(0.0143)	(0.0209)	(0.0156)	(0.0142)	(0.0145)	(0.0143)	(0.0144)
[p-val]		[6.17e-40]	[1.09e-19]	[4e-74]	[2.06e-102]	[7.8e-78]	[0.00156]	[4.66e-53]
RMSE_sum	11.39	**11.38**	15.01	16.43	18.62	18.52	11.4	11.4
(sd)	(1.26)	(1.26)	(1.48)	(1.57)	(2.04)	(1.98)	(1.26)	(1.27)
[p-val]		[0.0284]	[1.29e-51]	[3.02e-59]	[1.6e-61]	[4.98e-62]	[0.0022]	[0.339]
CRPS_sum	5.978	**5.97**	8.845	10.05	11.26	11.26	5.978	5.991
(sd)	(0.597)	(0.598)	(1.04)	(1.17)	(1.38)	(1.34)	(0.596)	(0.599)
[p-val]		[0.0442]	[2.51e-56]	[9.07e-64]	[1.19e-64]	[1.08e-65]	[0.714]	[0.0731]
RMSE_F	0.1071	**0.046**		0.1654	0.4385	0.2946	0.1071	0.1752
(sd)	(0.0139)	(0.0253)		(0.0131)	(0.00974)	(0.0144)	(0.014)	(0.0107)
[p-val]		[2.55e-63]		[1.58e-76]	[1.95e-139]	[8.68e-103]	[0.958]	[4.38e-94]
RMSE_b	0.3068	**0.3054**		0.6919	0.3323	0.3171	0.3067	0.3094
(sd)	(0.00975)	(0.00969)		(0.0215)	(0.00953)	(0.0103)	(0.00972)	(0.00972)
[p-val]		[1.12e-20]		[8.44e-122]	[3.8e-63]	[1.71e-48]	[0.0133]	[1.27e-24]
sigma2	0.03031	0.0215	0.7859	0.3629	0.1563	0.03185	0.02317	0.09467
sigma2_b	0.06985	0.02572			0.07174	0.07143	0.03383	0.07061
time (s)	0.4429	0.003952	0.03601	8.319	293.9	0.5796	1.99	0.07482

Table 6: Results for the grouped random effects and the mean function F = 'linear'. For all accuracy metrics (RMSE, CRPS, quantile loss), averages over the simulatiom runs are reported. Corresponding standard deviations are in parentheses. P-values are calculated using paired t-tests comparing the GPBoost algorithm to the other approaches. For the (co-)variance parameters, RMSEs are reported. 'GBPOOS' refers to the GPBoostOOS algorithm and 'GPBFixLR' denotes the GPBoost algorithm with the learning rate held fixed at 0.1. 'QL' denotes the quantile loss. Results for the "extrapolation" data sets, see Section 4.1, are denoted by '_ext' and results for the predictions of sums are denoted by '_sum'. The smallest values (excluding 'GBPOOS' and 'GPBFixLR') are in boldface. An empty value indicates that the required predictions or estimates cannot be calculated.

	GPBoost	LinearGP	LSBoost	mboost	GPBoostOOS	GPBFixLR
RMSE	**1.359**	1.466	1.479	1.69	1.358	1.36
(sd)	(0.0538)	(0.0572)	(0.0621)	(0.0846)	(0.0549)	(0.0534)
[p-val]		[1.69e-60]	[1.1e-43]	[3.76e-66]	[0.458]	[0.0708]
CRPS	**0.783**	0.8183	0.8512	0.9488	0.7607	0.7838
(sd)	(0.0357)	(0.0298)	(0.0402)	(0.0479)	(0.0296)	(0.0355)
[p-val]		[2.17e-29]	[1.33e-38]	[1.82e-57]	[8.51e-25]	[0.15]
RMSE_ext	**1.505**	1.6	1.591	107900	1.504	1.507
(sd)	(0.0934)	(0.0897)	(0.14)	(541000)	(0.0928)	(0.0941)
[p-val]		[1.77e-46]	[8.31e-15]	[0.0489]	[0.457]	[0.0403]
CRPS_ext	**0.8576**	0.897	0.9289	107900	0.8456	0.859
(sd)	(0.0603)	(0.0518)	(0.0987)	(541000)	(0.0532)	(0.0606)
[p-val]		[4.18e-29]	[1.41e-17]	[0.0489]	[3.11e-16]	[0.00797]
RMSE_sum	**11.6**	11.95	13.53	1525000	11.6	11.62
(sd)	(2.07)	(2.03)	(2.87)	(7650000)	(2.06)	(2.07)
[p-val]		[4.1e-07]	[8.33e-16]	[0.0489]	[0.826]	[0.262]
CRPS_sum	**6.376**	6.533	8.082	1079000	6.311	6.381
(sd)	(1.07)	(1.02)	(1.9)	(5410000)	(1.01)	(1.07)
[p-val]		[4.31e-05]	[1.18e-20]	[0.0489]	[0.00202]	[0.474]
RMSE_F	**0.6837**	0.8748			0.6822	0.6858
(sd)	(0.0821)	(0.069)			(0.0824)	(0.0829)
[p-val]		[6.81e-63]			[0.175]	[0.0301]
RMSE_b	**0.7033**	0.7104			0.7028	0.7026
(sd)	(0.071)	(0.0641)			(0.0707)	(0.0716)
[p-val]		[0.0378]			[0.804]	[0.0761]
sigma2	0.5169	0.659	0.1183	1.847	0.5137	0.5141
sigma2_b	0.258	0.327			0.3121	0.2577
rho	0.03292	0.04431			0.04289	0.03271
time (s)	5.251	0.5671	0.02366	0.7172	13.99	4.265

Table 7: Results for the Gaussian process and the mean function F = 'hajjem'. For all accuracy metrics (RMSE, CRPS, quantile loss), averages over the simulatiom runs are reported. Corresponding standard deviations are in parentheses. P-values are calculated using paired t-tests comparing the GPBoost algorithm to the other approaches. For the (co-)variance parameters, RMSEs are reported. 'GBPOOS' refers to the GPBoostOOS algorithm and 'GPBFixLR' denotes the GPBoost algorithm with the learning rate held fixed at 0.1. 'QL' denotes the quantile loss. Results for the "extrapolation" data sets, see Section 4.1, are denoted by '_ext' and results for the predictions of sums are denoted by '_sum'. The smallest values (excluding 'GBPOOS' and 'GPBFixLR') are in boldface. An empty value indicates that the required predictions or estimates cannot be calculated.

	GPBoost	LinearGP	LSBoost	mboost	GPBoostOOS	GPBFixLR
RMSE	1.203	**1.184**	1.353	1.475	1.202	1.245
(sd)	(0.042)	(0.039)	(0.0672)	(0.0803)	(0.0426)	(0.046)
[p-val]		[1.63e-28]	[2.8e-54]	[5.62e-65]	[0.0263]	[4.64e-45]
CRPS	0.6798	**0.6681**	0.7652	0.8342	0.6786	0.7082
(sd)	(0.0245)	(0.0218)	(0.0385)	(0.0461)	(0.0239)	(0.0287)
[p-val]		[4.23e-27]	[2.39e-54]	[6.87e-64]	[8.58e-05]	[9.79e-46]
RMSE_ext	1.391	**1.373**	1.454	519100	1.391	1.426
(sd)	(0.0968)	(0.0972)	(0.107)	(1310000)	(0.0973)	(0.0982)
[p-val]		[2.11e-16]	[3.21e-10]	[0.000144]	[0.911]	[4.99e-37]
CRPS_ext	0.787	**0.7765**	0.8268	519100	0.7864	0.8106
(sd)	(0.0575)	(0.0566)	(0.0667)	(1310000)	(0.0562)	(0.0604)
[p-val]		[4.08e-16]	[1.78e-10]	[0.000144]	[0.158]	[2.96e-39]
RMSE_sum	11.43	**11.32**	13.72	7341000	11.45	11.59
(sd)	(1.97)	(1.98)	(2.1)	(18600000)	(1.98)	(1.99)
[p-val]		[0.00172]	[4.08e-20]	[0.000144]	[0.24]	[6.8e-07]
CRPS_sum	6.143	**6.075**	7.966	5191000	6.148	6.278
(sd)	(0.96)	(0.962)	(1.32)	(13100000)	(0.962)	(0.988)
[p-val]		[0.000119]	[3.23e-28]	[0.000144]	[0.529]	[7.49e-11]
RMSE_F	0.3187	**0.2093**			0.3145	0.4492
(sd)	(0.102)	(0.129)			(0.103)	(0.0878)
[p-val]		[3.06e-30]			[3.57e-08]	[1.44e-57]
RMSE_b	0.6658	**0.6607**			0.6665	0.6745
(sd)	(0.0584)	(0.0556)			(0.0592)	(0.0601)
[p-val]		[0.0527]			[0.179]	[3.52e-10]
sigma2	0.1663	0.1196	0.6159	1.016	0.1427	0.3356
sigma2_b	0.2286	0.2427			0.249	0.2416
rho	0.03604	0.03877			0.03949	0.03246
time (s)	9.445	0.5274	0.01511	0.7721	21.25	3.287

Table 8: Results for the Gaussian process and the mean function F = 'linear'. For all accuracy metrics (RMSE, CRPS, quantile loss), averages over the simulatiom runs are reported. Corresponding standard deviations are in parentheses. P-values are calculated using paired t-tests comparing the GPBoost algorithm to the other approaches. For the (co-)variance parameters, RMSEs are reported. 'GBPOOS' refers to the GPBoostOOS algorithm and 'GPBFixLR' denotes the GPBoost algorithm with the learning rate held fixed at 0.1. 'QL' denotes the quantile loss. Results for the "extrapolation" data sets, see Section 4.1, are denoted by '_ext' and results for the predictions of sums are denoted by '_sum'. The smallest values (excluding 'GBPOOS' and 'GPBFixLR') are in boldface. An empty value indicates that the required predictions or estimates cannot be calculated.