Mach Learn
DOI 10.1007/s10994-016-5597-1

Boosted multivariate trees for longitudinal data

Amol Pande[1] · Liang Li[2] · Jeevanantham Rajeswaran[3] · John Ehrlinger[3] · Udaya B. Kogalur[3] · Eugene H. Blackstone[4] · Hemant Ishwaran[1]

Received: 12 April 2016 / Accepted: 18 October 2016

Abstract Machine learning methods provide a powerful approach for analyzing longitudinal data in which repeated measurements are observed for a subject over time. We boost multivariate trees to fit a novel flexible semi-nonparametric marginal model for longitudinal data. In this model, features are assumed to be nonparametric, while feature-time interactions are modeled semi-nonparametrically utilizing P-splines with estimated smoothing parameter. In order to avoid overfitting, we describe a relatively simple in sample cross-validation method which can be used to estimate the optimal boosting iteration and which has the surprising added benefit of stabilizing certain parameter estimates. Our new multivariate tree boosting method is shown to be highly flexible, robust to covariance misspecification and unbalanced designs, and resistant to overfitting in high dimensions. Feature selection can be used to identify important features and feature-time interactions. An application to longitudinal data of forced 1-second lung expiratory volume (FEV1) for lung transplant patients identifies an important feature-time interaction and illustrates the ease with which our method can find complex relationships in longitudinal data.

Keywords Gradient boosting · Marginal model · Multivariate regression tree · P-splines · Smoothing parameter

Editor: Hendrik Blockeel.

✉ Hemant Ishwaran
hemant.ishwaran@gmail.com

1 Division of Biostatistics, University of Miami, Miami, FL, USA

2 The University of Texas MD Anderson Cancer Center, Houston, TX, USA

3 Department of Quantitative Health Sciences, Cleveland Clinic, Cleveland, OH, USA

4 Department of Heart and Vascular Institute, Cleveland Clinic, Cleveland, OH, USA

Published online: 04 November 2016

1 Introduction

The last decade has witnessed a growing use of machine learning methods in place of traditional statistical approaches as a way to model the relationship between the response and features. Boosting is one of the most successful of these machine learning methods. It was originally designed for classification problems (Freund and Schapire 1996), but later successfully extended to other settings such as regression and survival problems. Recent work has also sought to extend boosting from univariate response settings to more challenging multivariate response settings, including longitudinal data. The longitudinal data scenario in particular offers many nuances and challenges unlike those in univariate response modeling. This is because in longitudinal data, the response for a given subject is measured repeatedly over time. Hence any optimization function that involves the conditional mean of the response must also take into account the dependence in the response values for a given subject. Furthermore, nonlinear relationships between features and the response may involve time.

An effective way to approach longitudinal data is through what is called the marginal model (Diggle et al. 2002). The marginal model provides a flexible means for estimating mean time-profiles without requiring a distributional model for the Y response, requiring instead only an assumption regarding the mean and the covariance. Formally, we assume the data is $\{(\mathbf{y}_i, \mathbf{t}_i, \mathbf{x}_i)\}_1^n$ where each subject i has $n_i \geq 1$ continuous response values $\mathbf{y}_i = (y_{i,1}, \ldots, y_{i,n_i})^T$ measured at possibly different time points $t_{i,1} \leq t_{i,2} \leq \ldots \leq t_{i,n_i}$ and $\mathbf{x}_i \in \mathbb{R}^p$ is the p-dimensional feature. To estimate the mean time-profile, the marginal model specifies the conditional mean $E(\mathbf{Y}_i|\mathbf{x}_i, \mathbf{t}_i) = \boldsymbol{\mu}_i$ under a variance assumption $\text{Var}(\mathbf{Y}_i|\mathbf{x}_i) = \mathbf{V}_i$. Typically, $\mathbf{V}_i = \phi\mathbf{R}_i$ where $\mathbf{R}_i$ is an $n_i \times n_i$ correlation matrix parameterized by a finite set of parameters and $\phi > 0$ is an unknown dispersion parameter.

The marginal model expresses the conditional mean $\boldsymbol{\mu}_i$ as a function of features and time. Typically in the statistical literature this function is specified parametrically as a linear combination of features and time. In most cases, linear functions can be very restrictive, and therefore various generalizations have been proposed to make the model more flexible and less susceptible to model misspecification. These include, for example, adding two-way cross-product interactions between features and time, using generalized additive models (Hastie and Tibshirani 1990) which allow for nonlinear feature or time effects, and time-varying coefficient models (Hoover et al. 1998). Some of these extensions (e.g., generalized additive models, time-varying coefficient models) are referred to as being semi-parametric because the overall structure of the model is parametric, but certain low-dimensional components are estimated nonparametrically as smooth functions. Although these models are more flexible compared with linear models, unless specified explicitly, these models do not allow for non-linear interactions among multiple features or non-linear interactions of multiple features and time.

To overcome these limitations of standard statistical modeling, researchers have turned increasingly to the use of boosting for longitudinal data. Most of these applications are based on the mixed effect models. For example, using likelihood-based boosting, Tutz and Reithinger (2007) described mixed effects modeling using semiparametric splines for fixed effects, while Groll and Tutz (2012) considered generalized additive models subject to P-splines (see Tutz and Binder 2006, for background on likelihood-based boosting). The R-package `mboost`, which implements boosting using additive base learners for univariate response (Hothorn et al. 2010, 2016), now includes random effect base learners to handle longitudinal data. This approach was used by Mayr et al. (2012) for quantile longitudinal

regression. All of these methods implement componentwise boosting where only one component is fit for a given boosting step (an exception is `mboost` which allows tree base learner for fitting multiple features simultaneously). Although componentwise boosting has proven particularly useful for high dimensional parametric settings, it is not well suited for nonparametric settings, especially if the goal is to nonparametrically model feature-time interactions and identify such effects using feature selection.

1.1 A semi-nonparametric multivariate tree boosting approach

In this article we boost multivariate trees to fit a flexible marginal model. This marginal model allows for nonlinear feature and time effects as well as nonlinear interactions among multiple features and time, and hence is more flexible than previous semiparametric models. For this reason, we have termed this more flexible approach "semi-nonparametric". Our model assumes the vector of mean values $\boldsymbol{\mu}_i = (\mu_{i,1}, \ldots, \mu_{i,n_i})^T$ satisfies

$$\mu_{i,j} = \beta_0(\mathbf{x}_i) + \sum_{l=1}^{d} b_l(t_{i,j})\beta_l(\mathbf{x}_i), \qquad j = 1, \ldots, n_i. \tag{1}$$

Here, β_0 and $\{\beta_l\}_1^d$ represent fully unspecified real-valued functions of $\mathbf{x}$ and $\{b_l\}_1^d$ are a collection of prespecified functions that map time to a desired basis and are used to model feature-time interactions. Examples of $\{b_l\}_1^q$ basis functions include the class of low-rank thin-plate splines (Duchon 1977; Wahba 1990), which correspond to semi-nonparametric models of the form

$$\mu_{i,j} = \beta_0(\mathbf{x}_i) + t_{i,j}\beta_1(\mathbf{x}_i) + \sum_{l=2}^{d} |t_{i,j} - \kappa_{l-1}|^{(2m-1)}\beta_l(\mathbf{x}_i), \tag{2}$$

where $\kappa_1 < \cdots < \kappa_{d-1}$ are prespecified knots. Another example are truncated power basis splines of degree q (Ruppert et al. 2003):

$$\mu_{i,j} = \beta_0(\mathbf{x}_i) + \sum_{l=1}^{q} t_{i,j}^l \beta_l(\mathbf{x}_i) + \sum_{l=q+1}^{d} (t_{i,j} - \kappa_{l-q})_+^q \beta_l(\mathbf{x}_i).$$

Another useful class of families are B-splines (De Boor 1978). In this manuscript we will focus exclusively on the class of B-splines.

According to (1), subjects with the same feature $\mathbf{x}$ have the same conditional mean trajectory for a given $\mathbf{t}$ as specified by a spline curve: the shape of the curve is altered by the spline coefficients, $\{\beta_l(\mathbf{x})\}_1^d$. Two specifications maximize the flexibility of (1). First, each spline coefficient is a nonparametric function of all p features (i.e., $\beta_l(.)$ is a scalar function with multivariate input). Second, similar to the penalized spline literature, we use a large number of basis functions to ensure the flexibility of the conditional mean trajectory (Ruppert et al. 2003). While (1) is in principle very general, it is worth pointing out that simpler, but still useful, models are accommodated within (1). For example, when $d = 1$ and $b_1(t_{i,j}) = t_{i,j}$, model (1) specializes to $\beta_0(\mathbf{x}_i) + \beta_1(\mathbf{x}_i)t_{i,j}$, which implies that given the baseline features $\mathbf{x}_i$, the longitudinal mean trajectory is linear with intercept $\beta_0(\mathbf{x}_i)$ and slope $\beta_1(\mathbf{x}_i)$. This model may be useful when there are a small number of repeated measures per subject. When both $\beta_0(\mathbf{x}_i)$ and $\beta_1(\mathbf{x}_i)$ are linear combinations of $\mathbf{x}_i$, the model reduces to a parametric longitudinal model with linear additive feature and linear two-way cross-product interactions between features and time.

Let $\boldsymbol{\beta}(\mathbf{x}) = (\beta_0(\mathbf{x}), \beta_1(\mathbf{x}), \ldots, \beta_d(\mathbf{x}))^T$ denote the vector of $(d+1)$-dimensional feature functions from (1). In this manuscript, we estimate $\boldsymbol{\beta}(\mathbf{x})$ nonparametrically by boosting multivariate regression trees, a method we call boostmtree. While there has been much recent interest in boosting longitudinal data, there has been no systematic attempt to boost multivariate trees in such settings. Doing so has many advantages, including that it allows us to accommodate non-linearity of features as well as non-linear interactions of multiple features without having to specify them explicitly. The boostmtree approach is an extension of Friedman's (2001) tree-based gradient boosting to multivariate responses. Section 2 describes this extension and presents a general framework for boosting longitudinal data using a generic (but differentiable) loss function. Section 3 builds upon this general framework to describe the boostmtree algorithm. There we introduce an ℓ_2-loss function which incorporates both the target mean structure (1) as well as a working correlation matrix for addressing dependence in response values.

The boostmtree algorithm presented in Sect. 3 represents a high-level description of the algorithm in that it assumes that parameters such as the correlation coefficient of the repeated measurements are fixed quantities. But in practice in order to increase the efficiency of boostmtree, we must estimate these additional parameters. In this manuscript, all parameters except $\{\boldsymbol{\mu}_i\}_1^n$ are referred to as ancillary parameters. Estimation of ancillary parameters are described in Sect. 4. This includes a simple update for the correlation matrix that can be implemented using standard software and which can accommodate many covariance models. We also present a simple method for estimating the smoothing parameter for penalizing the semiparametric functions $\{b_l\}_1^d$. This key feature allows flexible nonparametric modeling of the feature space while permitting smoothed, penalized spline-based time-feature estimates. In addition, in order to determine an optimal boosting step, we introduce a novel "in sample" cross-validation method. In boosting, the optimized number of boosting iterations is traditionally determined using cross-validation, but this can be computationally intensive for longitudinal data. The new in sample method alleviates this problem and has the added benefit that it stabilizes the working correlation estimator which suffers from a type of rebound effect without this. The unintended consequence of introducing instability while estimating an ancillary parameter is a new finding we believe, and may be applicable in general to any boosting procedure where ancillary parameters are estimated outside of the main boosting procedure. The in sample method we propose may provide a general solution for addressing this subtle issue.

Computational tractability is another important feature of boostmtree. By using multivariate trees, the matching pursuit approximation is reduced to calculating a small collection of weighted generalized ridge regression estimators. The ridge component is induced by the penalization of the basis functions and thus penalization serves double duty here. It not only helps to avoid overfitting, but it also numerically stabilizes the boosted estimator. This makes boostmtree robust to design specifications. In Sect. 5, we investigate performance of boostmtree using simulations. Performance is assessed in terms of prediction and feature selection accuracy. We compare boostmtree to several boosting procedures. Even when some of these models are specified to match the data generating mechanism, we find boostmtree does nearly as well, while in complex settings it generally outperforms other methods. We also find that boostmtree performs very well in terms of feature selection. Without explicitly specifying the relationship of response with features and time, we are able to select important features, but also separate features that affect the response directly from those that affect the response through time interactions. In Sect. 6, we use boostmtree to analyze longitudinal data of forced 1-second lung expiratory volume (FEV1) for lung transplant patients. We evaluate the temporal trend of FEV1 after transplant, and identify factors predictive of FEV1 and

assess differences in time-profile trends for single versus double lung transplant patients. Section 7 discusses our overall findings.

2 Gradient multivariate tree boosting for longitudinal data

Friedman (2001) introduced gradient boosting, a general template for applying boosting. The method has primarily been applied to univariate settings, but can be extended to multivariate longitudinal settings as follows. We assume a generic loss function, denoted by L. Let $(\mathbf{y}, \mathbf{t}, \mathbf{x})$ denote a generic data point. We assume

$$E(\mathbf{Y}|\mathbf{X} = \mathbf{x}) = \boldsymbol{\mu}(\mathbf{x}) = F(\boldsymbol{\beta}(\mathbf{x}))$$

where F is a known function that can depend on $\mathbf{t}$. A key assumption used later in our development is that F is assumed to be a linear operator. As described later, F will correspond to the linear operator obtained by expanding spline-basis functions over time in model (1).

In the framework described in Friedman (2001), the goal is to boost the predictor $F(\mathbf{x})$, but because our model is parameterized in terms of $\boldsymbol{\beta}(\mathbf{x})$, we boost this function instead. Our goal is to estimate $\boldsymbol{\beta}(\mathbf{x})$ by minimizing $E\left[L(\mathbf{Y}, F(\boldsymbol{\beta}(\mathbf{x})))\right]$ over some suitable space. Gradient boosting applies a stagewise fitting procedure to provide an approximate solution to the target optimization. Thus starting with an initial value $\boldsymbol{\beta}^{(0)}(\mathbf{x})$, the value at iteration $m = 1, \ldots, M$ is updated from the previous value according to

$$\boldsymbol{\beta}^{(m)}(\mathbf{x}) = \boldsymbol{\beta}^{(m-1)}(\mathbf{x}) + \nu \mathbf{h}(\mathbf{x}; \mathbf{a}_m), \qquad \boldsymbol{\mu}^{(m)}(\mathbf{x}) = F(\boldsymbol{\beta}^{(m)}(\mathbf{x})).$$

Here $0 < \nu \leq 1$ is a learning parameter while $\mathbf{h}(\mathbf{x}; \mathbf{a}) \in \mathbb{R}^{d+1}$ denotes a base learner parameterized by the value $\mathbf{a}$. The notation $\mathbf{h}(\mathbf{x}; \mathbf{a}_m)$ denotes the optimized base learner where optimization is over $\mathbf{a} \in \mathscr{A}$, where $\mathscr{A}$ represents the set of parameters of the weak learner. Typically, a small value of ν is used, say $\nu = 0.05$, which has the effect of slowing the learning of the boosting procedure and therefore acts a regularization mechanism.

One method for optimizing the base learner is by solving the matching pursuit problem (Mallat and Zhang 1993):

$$\mathbf{a}_m = \underset{\mathbf{a} \in \mathscr{A}}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(\mathbf{y}_i, \boldsymbol{\mu}_i^{(m-1)} + F(\mathbf{h}(\mathbf{x}_i; \mathbf{a}))\right).$$

Because solving the above may not always be easy, gradient boosting instead approximates the matching pursuit problem with a two-stage procedure: (i) find the base learner closest to the gradient in an ℓ_2-sense; (ii) solve a one-dimensional line-search problem.

The above extension which assumes a fixed loss function addresses simpler longitudinal settings, such as balanced designs. To accommodate more general settings we must allow the loss function to depend on i. This is in part due to the varying sample size n_i, which alters the dimension of the response, and hence affects the loss function, but also because we must model the correlation, which may also depend on i. Therefore, we will denote the loss function by L_i to indicate its dependence on i. This subscript i notation will be used throughout in general to identify any term which may depend on i. In particular, since the mean may also in general depend upon i, since it depends upon the observed time points, we will write

$$E(\mathbf{Y}_i|\mathbf{X}_i = \mathbf{x}_i) = \boldsymbol{\mu}_i(\mathbf{x}_i) = F_i(\boldsymbol{\beta}(\mathbf{x}_i)). \tag{3}$$

In this more general framework, the matching pursuit problem becomes

$$\mathbf{a}_m = \underset{\mathbf{a}\in\mathscr{A}}{\operatorname{argmin}} \sum_{i=1}^{n} L_i\left(\mathbf{y}_i, \boldsymbol{\mu}_i^{(m-1)} + F_i(\mathbf{h}(\mathbf{x}_i;\mathbf{a}))\right).$$

We use multivariate regression trees for the base learner and approximate the above matching pursuit problem using the following two-stage gradient boosting approach. Let the negative gradient for subject i with respect to $\boldsymbol{\beta}(\mathbf{x}_i)$ evaluated at $\boldsymbol{\beta}^{(m-1)}(\mathbf{x}_i)$ be

$$\mathbf{g}_{m,i} = -\left.\frac{\partial L_i(\mathbf{y}_i, \boldsymbol{\mu}_i)}{\partial \boldsymbol{\beta}(\mathbf{x}_i)}\right|_{\boldsymbol{\beta}(\mathbf{x}_i)=\boldsymbol{\beta}^{(m-1)}(\mathbf{x}_i)}.$$

To determine the ℓ_2-closest base learner to the gradient, we fit a K-terminal node multivariate regression tree using $\{\mathbf{g}_{m,i}\}_1^n$ for the responses and $\{\mathbf{x}_i\}_1^n$ as the features, where $K \geq 1$ is prespecified value. Denote the resulting tree by $\mathbf{h}(\mathbf{x}; \{R_{k,m}\}_1^K)$, where $R_{k,m}$ represents the kth terminal node of the regression tree. Letting $\mathbf{f}_k \in \mathbb{R}^{d+1}$ denote the kth terminal node mean value, the ℓ_2-optimized base learner is

$$\mathbf{h}(\mathbf{x}; \{R_{k,m}\}_1^K) = \sum_{k=1}^{K} \mathbf{f}_k 1\left(\mathbf{x} \in R_{k,m}\right), \qquad \mathbf{f}_k = \frac{1}{|R_{k,m}|} \sum_{\mathbf{x}_i \in R_{k,m}} \mathbf{g}_{m,i}.$$

This completes the first step in the gradient boosting approximation. The second step typically involves a line search; however in univariate tree-based boosting (Friedman 2001, 2002), the line search is replaced with a more refined estimate which replaces the single line search parameter with a unique value for each terminal node. In the extension to multivariate trees, we replace $\{\mathbf{f}_k\}_1^K$ with $(d+1)$-vector valued estimates $\{\boldsymbol{\gamma}_{k,m}\}_1^K$ determined by optimizing the loss function

$$\{\boldsymbol{\gamma}_{k,m}\}_1^K = \underset{\{\boldsymbol{\gamma}_k\}_1^K}{\operatorname{argmin}} \sum_{i=1}^{n} L_i\left(\mathbf{y}_i, \boldsymbol{\mu}_i^{(m-1)} + F_i\left(\sum_{k=1}^{K} \boldsymbol{\gamma}_k 1\left(\mathbf{x}_i \in R_{k,m}\right)\right)\right).$$

The optimized base learner parameter is $\mathbf{a}_m = \{(R_{k,m}, \boldsymbol{\gamma}_{k,m})\}_1^K$ and the optimized learner is $\mathbf{h}(\mathbf{x}; \mathbf{a}_m) = \sum_{k=1}^K \boldsymbol{\gamma}_{k,m} 1\left(\mathbf{x} \in R_{k,m}\right)$. Because the terminal nodes $\{R_{k,m}\}_1^K$ of the tree form a partition of the feature space, the optimization of the loss function can be implemented one parameter at a time, thereby greatly simplifying computations. It is easily shown that

$$\boldsymbol{\gamma}_{k,m} = \underset{\boldsymbol{\gamma}_k \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \sum_{\mathbf{x}_i \in R_{k,m}} L_i\left(\mathbf{y}_i, \boldsymbol{\mu}_i^{(m-1)} + F_i(\boldsymbol{\gamma}_k)\right), \qquad k = 1, \ldots, K. \tag{4}$$

This leads to the following generic algorithm for boosting multivariate trees for longitudinal data; see Algorithm 1.

3 The boostmtree algorithm

Algorithm 1 describes a general template for boosting longitudinal data. We now use this to describe the boostmtree algorithm for fitting (1).

Algorithm 1 *Generic multivariate boosted trees for longitudinal data*

1: Initialize $\boldsymbol{\beta}^{(0)}(\mathbf{x}_i) = \mathbf{0}$, $\boldsymbol{\mu}_i^{(0)} = F_i(\mathbf{0})$, for $i = 1, \ldots, n$.
2: **for** $m = 1, \ldots, M$ **do**
3: $\mathbf{g}_{m,i} = -\left.\dfrac{\partial L_i(\mathbf{y}_i, \boldsymbol{\mu}_i)}{\partial \boldsymbol{\beta}(\mathbf{x}_i)}\right|_{\boldsymbol{\beta}(\mathbf{x}_i)=\boldsymbol{\beta}^{(m-1)}(\mathbf{x}_i)}, \qquad i = 1, \ldots, n.$
4: Fit a multivariate regression tree $\mathbf{h}(\mathbf{x}; \{R_{k,m}\}_1^K)$ using $\{(\mathbf{g}_{m,i}, \mathbf{x}_i)\}_1^n$ for data.
5: $\boldsymbol{\gamma}_{k,m} = \text{argmin}_{\boldsymbol{\gamma}_k \in \mathbb{R}^{d+1}} \sum_{\mathbf{x}_i \in R_{k,m}} L_i\left(\mathbf{y}_i, \boldsymbol{\mu}_i^{(m-1)} + F_i(\boldsymbol{\gamma}_k)\right), \qquad k = 1, \ldots, K.$
6: Update:

$$\boldsymbol{\beta}^{(m)}(\mathbf{x}) = \boldsymbol{\beta}^{(m-1)}(\mathbf{x}) + \nu \sum_{k=1}^{K} \boldsymbol{\gamma}_{k,m} 1(\mathbf{x} \in R_{k,m})$$

$$\boldsymbol{\mu}_i^{(m)}(\mathbf{x}) = F_i(\boldsymbol{\beta}^{(m)}(\mathbf{x})), \qquad i = 1, \ldots, n.$$

7: **end for**
8: Return $\left\{(\boldsymbol{\beta}^{(M)}(\mathbf{x}_i), \boldsymbol{\mu}_i^{(M)})_1^n\right\}$.

3.1 Loss function and gradient

We begin by defining the loss function required to calculate the gradient function. Assuming $\boldsymbol{\mu}_i$ as in (1), and denoting $\mathbf{V}_i$ for the working covariance matrix, where for the moment we assume $\mathbf{V}_i$ is known, the loss function is defined as follows

$$L_i(\mathbf{y}_i, \boldsymbol{\mu}_i) = (\mathbf{y}_i - \boldsymbol{\mu}_i)^T \mathbf{V}_i^{-1} (\mathbf{y}_i - \boldsymbol{\mu}_i)/2.$$

This can been seen to be an ℓ_2-loss function and in fact is often called the squared Mahalanobis distance. It is helpful to rewrite the covariance matrix as $\mathbf{V}_i = \phi \mathbf{R}_i$, where $\mathbf{R}_i$ represents the correlation matrix and ϕ a dispersion parameter. Because ϕ is a nuisance parameter unnecessary for calculating the gradient, we can remove it from our calculations. Therefore without loss of generality, we can work with the simpler loss function

$$L_i(\mathbf{y}_i, \boldsymbol{\mu}_i) = (\mathbf{y}_i - \boldsymbol{\mu}_i)^T \mathbf{R}_i^{-1} (\mathbf{y}_i - \boldsymbol{\mu}_i)/2.$$

We introduce the following notation. Let $\mathbf{D}_i = [\mathbf{1}_i, \mathbf{b}_1(\mathbf{t}_i), \ldots, \mathbf{b}_d(\mathbf{t}_i)]_{n_i \times (d+1)}$ denote the $n_i \times (d+1)$ design matrix for subject i where $\mathbf{1}_i = (1, \ldots, 1)_{n_i \times 1}^T$ and $\mathbf{b}_l(\mathbf{t}_i)$ is the expansion of $\{b_l\}_1^d$ over $\{\mathbf{t}_i\}_1^n$ evaluated at $\mathbf{t}_i$. Model (1) becomes

$$\boldsymbol{\mu}_i = \mathbf{D}_i \boldsymbol{\beta}(\mathbf{x}_i) = \mathbf{D}_i \begin{pmatrix} \beta_0(\mathbf{x}_i) \\ \vdots \\ \beta_d(\mathbf{x}_i) \end{pmatrix} = \beta_0(\mathbf{x}_i)\mathbf{1}_i + \sum_{l=1}^{d} \mathbf{b}_l(\mathbf{t}_i)\beta_l(\mathbf{x}_i). \tag{5}$$

Comparing (5) with (3) identifies the F_i in Algorithm 1 as

$$F_i(\boldsymbol{\beta}) = \mathbf{D}_i \boldsymbol{\beta}.$$

Hence, F_i is a linear operator on $\boldsymbol{\beta}$ obtained by expanding spline-basis functions over time. Working with a linear operator greatly simplifies calculating the gradient. The negative gradient for subject i with respect to $\boldsymbol{\beta}(\mathbf{x}_i)$ evaluated at the current estimator $\boldsymbol{\beta}^{(m-1)}(\mathbf{x}_i)$ is

$$\mathbf{g}_{m,i} = -\left.\frac{\partial L_i(\mathbf{y}_i, \boldsymbol{\mu}_i)}{\partial \boldsymbol{\beta}(\mathbf{x}_i)}\right|_{\boldsymbol{\beta}(\mathbf{x}_i)=\boldsymbol{\beta}^{(m-1)}(\mathbf{x}_i)} = \mathbf{D}_i^T \mathbf{R}_i^{-1} \left(\mathbf{y}_i - \boldsymbol{\mu}_i^{(m-1)}\right).$$

Upon fitting a multivariate regression tree to $\{(\mathbf{g}_{m,i}, \mathbf{x}_i)\}_1^n$, we must solve for $\boldsymbol{\gamma}_{k,m}$ in (4) where $F_i(\boldsymbol{\gamma}_k) = \mathbf{D}_i\boldsymbol{\gamma}_k$. This yields the weighted least squares problem

$$\left[\sum_{\mathbf{x}_i \in R_{k,m}} \mathbf{D}_i^T \mathbf{R}_i^{-1} \mathbf{D}_i\right] \boldsymbol{\gamma}_{k,m} = \sum_{\mathbf{x}_i \in R_{k,m}} \mathbf{g}_{m,i}. \tag{6}$$

3.2 Penalized basis functions

We utilize B-splines in (5). For flexible modeling a large number of knots are used which are subject to penalization to avoid overfitting. Penalization is implemented using the differencing penalty described in Eilers and Marx (1996). Penalized B-splines subject to this penalization are referred to as P-splines.

As the update to $\boldsymbol{\beta}(\mathbf{x})$ depends on $(\boldsymbol{\gamma}_{k,m})_1^K$, we impose P-spline regularization by penalizing $\boldsymbol{\gamma}_{k,m}$. Write $\boldsymbol{\gamma}_k = (\gamma_{k,1}, \ldots, \gamma_{k,d+1})^T$ for $k = 1, \ldots, K$. We replace (4) with the penalized optimization problem

$$\boldsymbol{\gamma}_{k,m} = \underset{\boldsymbol{\gamma}_k \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \left\{ \sum_{\mathbf{x}_i \in R_{k,m}} L_i\left(\mathbf{y}_i, \boldsymbol{\mu}_i^{(m-1)} + \mathbf{D}_i\boldsymbol{\gamma}_k\right) + \frac{\lambda}{2} \sum_{l=s+2}^{d+1} (\Delta_s \gamma_{k,l})^2 \right\}. \tag{7}$$

Here $\lambda \geq 0$ is a smoothing parameter and Δ_s denotes the $s \geq 1$ integer difference operator (Eilers and Marx 1996); e.g., for $s = 2$ the difference operator is defined by $\Delta_2\gamma_{k,l} = \Delta\left(\Delta\gamma_{k,l}\right) = \gamma_{k,l} - 2\gamma_{k,l-1} + \gamma_{k,l-2}$, for $l \geq 4 = s + 2$.

The optimization problem (7) can be solved by taking the derivative and solving for zero. Because the first coordinate of $\boldsymbol{\gamma}_k$ is unpenalized it will be convenient to decompose $\boldsymbol{\gamma}_k$ into the unpenalized first coordinate $\gamma_{k,1}$ and remaining penalized coordinates $\boldsymbol{\gamma}_k^{(2)} = (\gamma_{k,2}, \ldots, \gamma_{k,d+1})^T$. The penalty term can be expressed as

$$\sum_{l=s+2}^{d+1} (\Delta_s \gamma_{k,l})^2 = \left(\boldsymbol{\Delta}_s \boldsymbol{\gamma}_k^{(2)}\right)^T \boldsymbol{\Delta}_s \boldsymbol{\gamma}_k^{(2)} = \boldsymbol{\gamma}_k^{(2)T} \boldsymbol{\Delta}_s^T \boldsymbol{\Delta}_s \boldsymbol{\gamma}_k^{(2)}, \tag{8}$$

where $\boldsymbol{\Delta}_s$ is the matrix representation of the difference operator Δ_s. Let $\mathbf{P}_s = \boldsymbol{\Delta}_s^T \boldsymbol{\Delta}_s$, then the derivative of (8) is $2\mathbf{B}_s\boldsymbol{\gamma}_k$, where

$$\mathbf{B}_s = \begin{bmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{P}_s \end{bmatrix}_{(d+1)\times(d+1)}.$$

Closed form solutions for $\mathbf{B}_s$ are readily computed. Taking the derivative and setting to zero, the solution to $\boldsymbol{\gamma}_{k,m}$ in (7) is the following weighted generalized ridge regression estimator

$$\left[\sum_{\mathbf{x}_i \in R_{k,m}} \mathbf{D}_i^T \mathbf{R}_i^{-1} \mathbf{D}_i + \lambda \mathbf{B}_s\right] \boldsymbol{\gamma}_{k,m} = \sum_{\mathbf{x}_i \in R_{k,m}} \mathbf{g}_{m,i}. \tag{9}$$

This is the penalized analog of (6).

Remark 1 Observe that the ridge matrix $\mathbf{B}_s$ appearing in (9) is induced due to the penalization. Thus, the imposed penalization serves double duty: it penalizes splines, thereby mitigating overfitting, but it also ridge stabilizes the boosting estimator $\boldsymbol{\gamma}_{k,m}$, thus providing stability. The latter property is important when the design matrix $\mathbf{D}_i$ is singular, or near singular; for example due to replicated values of time, or due to a small number of time measurements.

Remark 2 We focus on penalized B-splines (P-splines) in this manuscript, but in principle our methodology can be applied to any other basis function as long as the penalization problem can be described in the form

$$\boldsymbol{\gamma}_{k,m} = \underset{\boldsymbol{\gamma}_k \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ \sum_{\mathbf{x}_i \in R_{k,m}} L_i \left(\mathbf{y}_i, \boldsymbol{\mu}_i^{(m-1)} + \sum_{j=1}^{2} \mathbf{D}_i^{(j)} \boldsymbol{\gamma}_k^{(j)} \right) + \lambda \boldsymbol{\gamma}_k^{(2)^T} \mathbf{P} \boldsymbol{\gamma}_k^{(2)} \right\} \tag{10}$$

where $\mathbf{P}$ is a positive definite symmetric penalty matrix. In (10), we have separated $\mathbf{D}_i$ into two matrices: the first matrix $\mathbf{D}_i^{(1)}$ equals the columns for the unpenalized parameters $\boldsymbol{\gamma}_k^{(1)}$, the second matrix $\mathbf{D}_i^{(2)}$ equals the remaining columns for the penalized parameters $\boldsymbol{\gamma}_k^{(2)}$ used for modeling the feature time-interaction effect. For example, for the class of thin-plate splines (2) with $m = 2$, one could use

$$\mathbf{D}_i^{(1)} = \left[1, t_{i,j}\right]_j, \quad \mathbf{D}_i^{(2)} = \left[|t_{i,j} - \kappa_1|^3, \ldots, |t_{i,j} - \kappa_{d-1}|^3\right]_j.$$

As reference, for the P-splines used here, $\mathbf{D}_i^{(1)} = \mathbf{1}_i, \mathbf{D}_i^{(2)} = [\mathbf{b}_1(\mathbf{t}_i), \ldots, \mathbf{b}_d(\mathbf{t}_i)]$, and $\mathbf{P} = \mathbf{P}_s$.

3.3 Boostmtree algorithm: fixed ancillary parameters

Combining the previous two sections, we arrive at the boostmtree algorithm which we have stated formally in Algorithm 2. Note that Algorithm 2 should be viewed as a high-level version of boostmtree in that it assumes a fixed correlation matrix and smoothing parameter. In Sect. 4, we discuss how these and other ancillary parameters can be updated on the fly within the algorithm. This leads to a more flexible boostmtree algorithm described later.

Algorithm 2 *Boostmtree (fixed ancillary parameters): A boosted semi-nonparametric marginal model using multivariate trees*

1: Initialize $\boldsymbol{\beta}^{(0)}(\mathbf{x}_i) = \mathbf{0}$, $\boldsymbol{\mu}_i^{(0)} = \mathbf{0}$, for $i = 1, \ldots, n$.
2: **for** $m = 1, \ldots, M$ **do**
3: $\quad \mathbf{g}_{m,i} = \mathbf{D}_i^T \mathbf{R}_i^{-1} \left(\mathbf{y}_i - \boldsymbol{\mu}_i^{(m-1)} \right)$.
4: $\quad$ Fit a multivariate regression tree $\mathbf{h}(\mathbf{x}; \{R_{k,m}\}_1^K)$ using $\{(\mathbf{g}_{m,i}, \mathbf{x}_i)\}_1^n$ for data.
5: $\quad$ Solve for $\boldsymbol{\gamma}_{k,m}$ in the weighted generalized ridge regression problem:

$$\left[\sum_{\mathbf{x}_i \in R_{k,m}} \mathbf{D}_i^T \mathbf{R}_i^{-1} \mathbf{D}_i + \lambda \mathbf{B}_s \right] \boldsymbol{\gamma}_{k,m} = \sum_{\mathbf{x}_i \in R_{k,m}} \mathbf{g}_{m,i}, \qquad k = 1, \ldots, K.$$

6: $\quad$ Update:

$$\boldsymbol{\beta}^{(m)}(\mathbf{x}) = \boldsymbol{\beta}^{(m-1)}(\mathbf{x}) + \nu \sum_{k=1}^{K} \boldsymbol{\gamma}_{k,m} 1(\mathbf{x} \in R_{k,m})$$

$$\boldsymbol{\mu}_i^{(m)}(\mathbf{x}) = \mathbf{D}_i \boldsymbol{\beta}^{(m)}(\mathbf{x}), \qquad i = 1, \ldots, n.$$

7: **end for**
8: Return $\left\{ \left(\boldsymbol{\beta}^{(M)}(\mathbf{x}_i), \boldsymbol{\mu}_i^{(M)} \right)_1^n \right\}$.

4 Estimating boostmtree ancillary parameters

In this section, we show how to estimate the working correlation matrix and the smoothing parameter as additional updates to the boostmtree algorithm. We also introduce an in sample CV method for estimating the number of boosting iterations and discuss an improved estimator for the correlation matrix based on the new in sample method. This will be shown to alleviate a "rebound" effect in which the boosted correlation rebounds back to zero due to overfitting.

4.1 Updating the working correlation matrix

As mentioned, Algorithm 2 assumed $\mathbf{R}_i$ was a fixed known quantity, however in practice $\mathbf{R}_i$ is generally unknown and must be estimated. Our strategy is to use the updated mean response to define a residual which is then fit using generalized least squares (GLS). We use GLS to estimate $\mathbf{R}_i$ from the fixed-effects intercept model

$$\mathbf{y}_i - \boldsymbol{\mu}_i^{(m)} = \alpha \mathbf{1}_i + \boldsymbol{\varepsilon}_i, \qquad i = 1, \ldots, n, \tag{11}$$

where $\mathrm{Var}(\boldsymbol{\varepsilon}_i) = \phi \mathbf{R}_i$. Estimating $\mathbf{R}_i$ under specified parametric models is straightforward using available software. We use the R-function `gls` from the `nlme` R-package (Pinheiro et al. 2014; Pinheiro and Bates 2000) and make use of the option `correlation` to select a parametric model for the working correlation matrix. Available working matrices include autoregressive processes of order 1 (`corAR1`), autoregressive moving average processes (`corARMA`), and exchangeable models (`corCompSymm`). Each are parameterized using only a few parameters, including a single unknown correlation parameter $-1 < \rho < 1$. In analyses presented later, we apply boostmtree using an exchangeable correlation matrix using `corCompSymm`.

4.2 Estimating the smoothing parameter

Algorithm 2 assumed a fixed smoothing parameter λ, but for greater flexibility we describe a method for estimating this value, λ_m, that can be implemented on the fly within the boostmtree algorithm. The estimation method exploits a well known trick of expressing an ℓ_2-optimization problem like (7) in terms of linear mixed models. First note that $\boldsymbol{\gamma}_{k,m}$ from (7) is equivalent to the best linear unbiased prediction estimator (BLUP estimator; Robinson (1991)) from the linear mixed model

$$\mathbf{y}_i = \boldsymbol{\mu}_i^{(m-1)} + \mathbf{X}_i \alpha_k + \mathbf{Z}_i \mathbf{u}_k + \boldsymbol{\varepsilon}_i, \quad i \in R_{k,m},$$

where

$$\begin{pmatrix} \mathbf{u}_k \\ \boldsymbol{\varepsilon}_i \end{pmatrix} \overset{\text{ind}}{\sim} \mathrm{N}\left(\begin{pmatrix} \mathbf{0} \\ \mathbf{0} \end{pmatrix}, \begin{bmatrix} \lambda^{-1} \mathbf{P}_s^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_i^{(m-1)} \end{bmatrix} \right)$$

and $\mathbf{R}_i^{(m-1)}$ denotes the current estimate for $\mathbf{R}_i$. In the above, α_k is the fixed effect corresponding to $\gamma_{k,1}$ with design matrix $\mathbf{X}_i = \mathbf{1}_i$, while $\mathbf{u}_k \in \mathbb{R}^d$ is the random effect corresponding to $\boldsymbol{\gamma}_k^{(2)}$ with $n_i \times d$ design matrix $\mathbf{Z}_i = [\mathbf{b}_1(\mathbf{t}_i), \ldots, \mathbf{b}_d(\mathbf{t}_i)]$. That is, each terminal node $R_{k,m}$ corresponds to a linear mixed model with a unique random effect $\mathbf{u}_k$ and fixed effect α_k.

Using the parameterization

$$\begin{aligned}
\tilde{\mathbf{y}}_i &= \left(\mathbf{R}_i^{(m-1)}\right)^{-1/2}\left(\mathbf{y}_i - \boldsymbol{\mu}_i^{(m-1)}\right)\\
\tilde{\mathbf{X}}_i &= \left(\mathbf{R}_i^{(m-1)}\right)^{-1/2}\mathbf{X}_i\\
\tilde{\mathbf{Z}}_i &= \left(\mathbf{R}_i^{(m-1)}\right)^{-1/2}\mathbf{Z}_i\mathbf{P}_s{}^{-1/2}\\
\tilde{\mathbf{u}}_k &= \mathbf{P}_s{}^{1/2}\mathbf{u}_k\\
\tilde{\boldsymbol{\varepsilon}}_i &= \left(\mathbf{R}_i^{(m-1)}\right)^{-1/2}\boldsymbol{\varepsilon}_i,
\end{aligned}$$

we obtain $\tilde{\mathbf{y}}_i = \tilde{\mathbf{X}}_i\alpha_k + \tilde{\mathbf{Z}}_i\tilde{\mathbf{u}}_k + \tilde{\boldsymbol{\varepsilon}}_i$, for $i \in R_{k,m}$, where

$$\begin{pmatrix}\tilde{\mathbf{u}}_k\\ \tilde{\boldsymbol{\varepsilon}}_i\end{pmatrix} \overset{\text{ind}}{\sim} \mathrm{N}\left(\begin{pmatrix}\mathbf{0}\\ \mathbf{0}\end{pmatrix}, \begin{bmatrix}\lambda^{-1}\mathbf{I}_d & \mathbf{0}\\ \mathbf{0} & \mathbf{I}_{n_i}\end{bmatrix}\right).$$

Perhaps the most natural way to estimate λ is to maximize the likelihood using restricted maximum likelihood estimation via mixed models. Combine the transformed data $\tilde{\mathbf{y}}_i$ across terminal nodes and apply a linear mixed model to the combined data; for example, by using mixed model software such as the `nlme` R-package (Pinheiro et al. 2014). As part of the model fitting this gives an estimate for λ.

While a mixed models approach may seem the most natural way to proceed, we have found in practice that the resulting computations are very slow, and only get worse with increasing sample sizes. Therefore we instead utilize an approximate, but computationally fast method of moments approach. Let $\tilde{\mathbf{X}}$, $\tilde{\mathbf{Z}}$ be the stacked matrices $\{\tilde{\mathbf{X}}_i\}_{i\in R_{k,m}}$, $\{\tilde{\mathbf{Z}}_i\}_{i\in R_{k,m}}$, $k = 1,\ldots,K$. Similarly, let $\boldsymbol{\alpha}$, $\tilde{\mathbf{u}}$, $\tilde{\mathbf{Y}}$, and $\tilde{\boldsymbol{\varepsilon}}$, be the stacked vectors for $\{\alpha_k\}_1^K$, $\{\tilde{\mathbf{u}}_k\}_1^K$, $\{\tilde{\mathbf{Y}}_i\}_{i\in R_{k,m}}$, and $\{\tilde{\boldsymbol{\varepsilon}}_i\}_{i\in R_{k,m}}$, $k = 1,\ldots,K$. We have

$$E\left[(\tilde{\mathbf{Y}} - \tilde{\mathbf{X}}\boldsymbol{\alpha})(\tilde{\mathbf{Y}} - \tilde{\mathbf{X}}\boldsymbol{\alpha})^T\right] = E\left[(\tilde{\mathbf{Z}}\tilde{\mathbf{u}} + \tilde{\boldsymbol{\varepsilon}})(\tilde{\mathbf{Z}}\tilde{\mathbf{u}} + \tilde{\boldsymbol{\varepsilon}})^T\right] = \lambda^{-1}\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T + E(\tilde{\boldsymbol{\varepsilon}}\tilde{\boldsymbol{\varepsilon}}^T).$$

This yields the following estimator:

$$\hat{\lambda} = \frac{\text{trace}(\tilde{\mathbf{Z}}\tilde{\mathbf{Z}}^T)}{\text{trace}[(\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\alpha})(\tilde{\mathbf{y}} - \tilde{\mathbf{X}}\boldsymbol{\alpha})^T] - N}, \qquad N = E(\tilde{\boldsymbol{\varepsilon}}^T\tilde{\boldsymbol{\varepsilon}}). \tag{12}$$

To calculate (12) requires a value for $\boldsymbol{\alpha}$. This we estimate using BLUP as follows. Fix $\hat{\lambda}$ at an initial value. The BLUP estimate $(\hat{\alpha}_k, \hat{\mathbf{u}}_k)$ for $(\alpha_k, \tilde{\mathbf{u}}_k)$ given $\hat{\lambda}$ are the solutions to the following set of equations (Robinson 1991):

$$\tilde{\mathbf{X}}^T\tilde{\mathbf{X}}\hat{\alpha}_k + \tilde{\mathbf{X}}^T\tilde{\mathbf{Z}}\hat{\mathbf{u}}_k = \tilde{\mathbf{X}}^T\tilde{\mathbf{y}}, \qquad \tilde{\mathbf{Z}}^T\tilde{\mathbf{X}}\hat{\alpha}_k + \left(\tilde{\mathbf{Z}}^T\tilde{\mathbf{Z}} + \hat{\lambda}\mathbf{I}\right)\hat{\mathbf{u}}_k = \tilde{\mathbf{Z}}^T\tilde{\mathbf{y}}. \tag{13}$$

Substituting the resulting BLUP estimate $\boldsymbol{\alpha} = \hat{\boldsymbol{\alpha}}$ into (12) yields an updated $\hat{\lambda}$. This process is repeated several times until convergence. Let λ_m be the final estimator. Now to obtain an estimate for $\boldsymbol{\gamma}_{k,m}$, we solve the following:

$$\left[\sum_{\mathbf{x}_i\in R_{k,m}} \mathbf{D}_i^T\left(\mathbf{R}_i^{(m-1)}\right)^{-1}\mathbf{D}_i + \lambda_m\mathbf{B}_s\right]\boldsymbol{\gamma}_{k,m} = \sum_{\mathbf{x}_i\in R_{k,m}} \mathbf{g}_{m,i}.$$

Remark 3 A stabler estimator for λ can be obtained by approximating N in place of using $N = E(\tilde{\boldsymbol{\varepsilon}}^T\tilde{\boldsymbol{\varepsilon}}) = \sum_i n_i$; the latter being implied by the transformed model. Let $\hat{\boldsymbol{\alpha}}$ and $\hat{\mathbf{u}}$ be

the current estimates for $\boldsymbol{\alpha}$ and $\tilde{\mathbf{u}}$. Approximate $\tilde{\boldsymbol{\varepsilon}}$ using the residual $\tilde{\boldsymbol{\varepsilon}}^* = \tilde{\mathbf{y}} - \tilde{\mathbf{X}}\hat{\boldsymbol{\alpha}} - \mathbf{Z}\hat{\mathbf{u}}$ and replace N with $\hat{N} = \tilde{\boldsymbol{\varepsilon}}^{*T}\tilde{\boldsymbol{\varepsilon}}^*$. This is the method used in the manuscript.

4.3 In sample cross-validation

In boosting, along with the learning parameter ν, the number of boosting steps M is also used as a regularization parameter in order to avoid overfitting. Typically the optimized value of M, denoted as M_{opt}, is estimated using either a hold-out test data or by using cross-validation (CV). But CV is computationally intensive, especially for longitudinal data. Information theoretic criteria such as AIC have the potential to alleviate this computational load. Successful implementation within the boosting paradigm is however fraught with challenges. Implementing AIC requires knowing the degrees of freedom of the fitted model which is difficult to do under the boosting framework. The degrees of freedom are generally underestimated which adversely affects estimation of M_{opt}. One solution is to correct the bias in the estimate of M_{opt} by using subsampling after AIC (Mayr et al. 2012). Such solutions are however applicable only to univariate settings. Applications of AIC to longitudinal data remains heavily underdeveloped with work focusing exclusively on parametric models within non-boosting contexts. For example, Pan (2001) described an extension of AIC to parametric marginal models. This replaces the traditional AIC degrees of freedom with a penalization term involving the covariance of the estimated regression coefficient. As this is a parametric regression approach, it cannot be applied to nonparametric models such as multivariate regression trees.

We instead describe a novel method for estimating M_{opt} that can be implemented within the boostmtree algorithm using a relatively simple, yet effective approach, we refer to as in sample CV. As before, let $R_{k,m}$ denote the kth terminal node of a boosted multivariate regression tree, where $k = 1, \ldots, K$. Assume that the terminal node for the ith subject is $R_{k_0,m}$ for some $1 \leq k_0 \leq K$. Let $R_{k_0,m,-i}$ be the new terminal node formed by removing i. Let λ_m be the current estimator of λ. Analogous to (7), we solve the following loss function within this new terminal node

$$\widetilde{\boldsymbol{\gamma}}^{(i)}_{k_0,m,-i} = \underset{\boldsymbol{\gamma}_k \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \left\{ \sum_{\mathbf{x}_j \in R_{k_0,m,-i}} L_j\left(\mathbf{y}_j, \widetilde{\boldsymbol{\mu}}_j^{(i,m-1)} + \mathbf{D}_j\boldsymbol{\gamma}_k\right) + \frac{\lambda_m}{2}\sum_{l=s+2}^{d+1}(\Delta_s\gamma_{k,l})^2 \right\}. \quad (14)$$

For each i, we maintain a set of n values $\{\widetilde{\boldsymbol{\mu}}_j^{(i,m-1)}\}_1^n$, where $\widetilde{\boldsymbol{\mu}}_j^{(i,m-1)}$ is the $(m-1)$th boosted in sample CV predictor for $\mathbf{y}_j$ treating i as a held out observation. The solution to (14) is used to update $\widetilde{\boldsymbol{\mu}}_j^{(i,m-1)}$ for those $\mathbf{x}_j$ in $R_{k_0,m}$. For those subjects that fall in a different terminal node $R_{k,m}$ where $k \neq k_0$, we use

$$\widetilde{\boldsymbol{\gamma}}^{(i)}_{k,m} = \underset{\boldsymbol{\gamma}_k \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \left\{ \sum_{\mathbf{x}_j \in R_{k,m}} L_j\left(\mathbf{y}_j, \widetilde{\boldsymbol{\mu}}_j^{(i,m-1)} + \mathbf{D}_j\boldsymbol{\gamma}_k\right) + \frac{\lambda_m}{2}\sum_{l=s+2}^{d+1}(\Delta_s\gamma_{k,l})^2 \right\}. \quad (15)$$

Once estimators (14) and (15) are obtained (a total of K optimization problems, each solved using weighted generalized ridge regression), we update $\widetilde{\boldsymbol{\mu}}_j^{(i,m-1)}$ for $j = 1, \ldots, n$ as follows:

$$\widetilde{\boldsymbol{\mu}}_j^{(i,m)} = \begin{cases} \widetilde{\boldsymbol{\mu}}_j^{(i,m-1)} + \nu\mathbf{D}_j\widetilde{\boldsymbol{\gamma}}^{(i)}_{k_0,m,-i} & \text{if } \mathbf{x}_j \in R_{k_0,m} \\ \widetilde{\boldsymbol{\mu}}_j^{(i,m-1)} + \nu\mathbf{D}_j\widetilde{\boldsymbol{\gamma}}^{(i)}_{k,m} & \text{if } \mathbf{x}_j \in R_{k,m} \text{ where } k \neq k_0. \end{cases}$$

Notice that $\widetilde{\boldsymbol{\mu}}_i^{(i,m)}$ represents the in sample CV predictor for $\mathbf{y}_i$ treating i as held out. Repeating the above for each $i = 1, \ldots, n$, we obtain $\{\widetilde{\boldsymbol{\mu}}_i^{(i,m)}\}_1^n$. We define our estimate of the root mean-squared error (RMSE) for the mth boosting iteration as

$$\widetilde{\mathrm{CV}}^{(m)} = \left[\frac{1}{n}\sum_{i=1}^{n}\frac{1}{n_i}\left(\mathbf{y}_i - \widetilde{\boldsymbol{\mu}}_i^{(i,m)}\right)^T\left(\mathbf{y}_i - \widetilde{\boldsymbol{\mu}}_i^{(i,m)}\right)\right]^{1/2}.$$

It is worth emphasizing that our approach has utilized all n subjects, rather than fitting a separate model using a subsample of the training data as done for CV. Therefore, the in sample CV can be directly incorporated into the boostmtree procedure to estimate M_{opt}. We also note that our method fits only one tree for each boosting iteration. For a true leave-one-out calculation, we should remove each observation i prior to fitting a tree and then solve the loss function. However, this is computationally intensive as it requires fitting n trees per iteration and solving nK weighted generalized ridge regressions. We have instead removed observation i from its terminal node as a way to reduce computations. Later we provide evidence showing the efficacy of this approach.

4.4 Rebound effect of the estimated correlation

Most of the applications of boosting are in the univariate setting where the parameter of interest is the conditional mean of the response. However in longitudinal studies, researchers are also interested in correctly estimating the correlation among responses for a given subject. We show that a boosting procedure whose primary focus is estimating the conditional mean of the response can be inefficient for estimating correlation without further modification. We show that by replacing $\boldsymbol{\mu}_i^{(m)}$ by $\widetilde{\boldsymbol{\mu}}_i^{(i,m)}$ in (11), an efficient estimate of correlation can be obtained.

Typically, gradient boosting tries to drive training error to zero. In boostmtree, this means that as the number of boosting iterations increases, the residual $\{\mathbf{y}_i - \boldsymbol{\mu}_i^{(m)}\}_1^n$ converges to zero in an ℓ_2-sense. The principle underlying the estimator (11) is to remove the effect of the true mean, so that the resulting residual values have zero mean and thereby making it relatively easy to estimate the covariance. Unfortunately, $\boldsymbol{\mu}_i^{(m)}$ not only removes the mean structure, but also the variance structure. This results in the estimated correlation having a rebound effect where the estimated value after attaining a maximum, will rebound and start a descent towards zero as m increases.

To see why this is so, consider an equicorrelation setting in which the correlation between responses for i are all equal to the same value $0 < \rho < 1$. By expressing $\boldsymbol{\varepsilon}_i$ from (11) as $\boldsymbol{\varepsilon}_i = b_i\mathbf{1}_i + \boldsymbol{\varepsilon}_i'$, we can rewrite (11) as the following random intercept model

$$\mathbf{y}_i - \boldsymbol{\mu}_i^{(m)} = \alpha\mathbf{1}_i + b_i\mathbf{1}_i + \boldsymbol{\varepsilon}_i', \qquad i = 1, \ldots, n. \tag{16}$$

The correlation between coordinates of $\mathbf{y}_i - \boldsymbol{\mu}_i^{(m)}$ equals $\rho = \sigma_b^2/(\sigma_b^2 + \sigma_e^2)$, where $\mathrm{Var}(b_i) = \sigma_b^2$ and $\mathrm{Var}(\boldsymbol{\varepsilon}_i') = \sigma_e^2\mathbf{I}_{n_i}$. In boostmtree, as the algorithm iterates, the estimate of ρ quickly reaches its optimal value. However, as the algorithm continues further, the residual $\mathbf{y}_i - \boldsymbol{\mu}_i^{(m)}$ decreases to zero in an ℓ_2-sense. This reduces the between subjects variation σ_b^2, which in turn reduces the estimate of ρ. As we show later, visually this represents a rebound effect of ρ.

On the other hand, notice that the in sample CV estimate $\widetilde{\boldsymbol{\mu}}_i^{(i,m)}$ described in the previous section is updated using all the subjects, except for subject i which is treated as being held out. This suggests a simple solution to the rebound effect. In place of $\mathbf{y}_i - \boldsymbol{\mu}_i^{(m)}$ for the

residual in (11), we use instead $\mathbf{y}_i - \widetilde{\boldsymbol{\mu}}_i^{(i,m)}$. The latter residual seeks to remove the effect of the mean but should not alter the variance structure as it does not converge to zero as m increases. Therefore, using this new residual should allow the correlation estimator to achieve its optimal value but will prevent the estimator from rebounding. Evidence of the effectiveness of this new estimator will be demonstrated shortly.

4.5 Boostmtree algorithm: estimated ancillary parameters

Combining the previous sections leads to Algorithm 3 given below which describes the boostmtree algorithm incorporating ancillary parameter updates for $\mathbf{R}_i$ and λ, and which includes the in sample CV estimator and corrected correlation matrix update.

Algorithm 3 *Boostmtree with estimated ancillary parameters*

Initialize $\boldsymbol{\beta}^{(0)}(\mathbf{x}_i) = \mathbf{0}$, $\boldsymbol{\mu}_i^{(0)} = \mathbf{0}$, $\mathbf{R}_i^{(0)} = \mathbf{I}_{n_i}$, for $i = 1, \ldots, n$.
for $m = 1, \ldots, M$ **do**
  $\mathbf{g}_{m,i} = \mathbf{D}_i^T \left(\mathbf{R}_i^{(m-1)}\right)^{-1} \left(\mathbf{y}_i - \boldsymbol{\mu}_i^{(m-1)}\right)$.
  Fit a multivariate regression tree $\mathbf{h}(\mathbf{x}; \{R_{k,m}\}_1^K)$ using $\{(\mathbf{g}_{m,i}, \mathbf{x}_i)\}_1^n$ for data.
  To estimate λ, cycle between (12) and (13) until convergence of $\hat{\lambda}$. Let λ_m denote the final estimator.
  Solve for $\boldsymbol{\gamma}_{k,m}$ in

$$\left[\sum_{\mathbf{x}_i \in R_{k,m}} \mathbf{D}_i^T \left(\mathbf{R}_i^{(m-1)}\right)^{-1} \mathbf{D}_i + \lambda_m \mathbf{B}_s\right] \boldsymbol{\gamma}_{k,m} = \sum_{\mathbf{x}_i \in R_{k,m}} \mathbf{g}_{m,i}, \qquad k = 1, \ldots, K.$$

  Update:

$$\boldsymbol{\beta}^{(m)}(\mathbf{x}) = \boldsymbol{\beta}^{(m-1)}(\mathbf{x}) + \nu \sum_{k=1}^{K} \boldsymbol{\gamma}_{k,m} 1(\mathbf{x} \in R_{k,m})$$

$$\boldsymbol{\mu}_i^{(m)}(\mathbf{x}) = \mathbf{D}_i \boldsymbol{\beta}^{(m)}(\mathbf{x}), \qquad i = 1, \ldots, n.$$

  if (in sample CV requested) **then**
    Update $\left\{\widetilde{\boldsymbol{\mu}}_i^{(i,m)}\right\}_1^n$ using (14) and (15). Calculate $\widetilde{\mathrm{CV}}^{(m)}$.
    Estimate $\mathbf{R}_i$ from (11), replacing $\boldsymbol{\mu}_i^{(m)}$ by $\widetilde{\boldsymbol{\mu}}_i^{(i,m)}$ and using `gls` under a parametric working correlation assumption. Update $\mathbf{R}_i^{(m)} \leftarrow \hat{\mathbf{R}}_i$ where $\hat{\mathbf{R}}_i$ is the resulting estimated value.
  else
    Estimate $\mathbf{R}_i$ from (11) using `gls` under a parametric working correlation assumption. Update $\mathbf{R}_i^{(m)} \leftarrow \hat{\mathbf{R}}_i$ where $\hat{\mathbf{R}}_i$ is the resulting estimated value.
  end if
end for
if (in sample CV requested) **then**
  Estimate M_{opt}
  Return $\left\{\left(\boldsymbol{\beta}^{(M_{\mathrm{opt}})}(\mathbf{x}_i), \boldsymbol{\mu}_i^{(M_{\mathrm{opt}})}\right)_1^n, M_{\mathrm{opt}}\right\}$.
else
  Return $\left\{\left(\boldsymbol{\beta}^{(M)}(\mathbf{x}_i), \boldsymbol{\mu}_i^{(M)}\right)_1^n\right\}$.
end if

5 Simulations and empirical results

We used three sets of simulations for assessing performance of boostmtree.

Simulation I The first simulation assumed the model:

$$\mu_{i,j} = C_0 + \sum_{k=1}^{4} C_k^* x_i^{*(k)} + \sum_{l=1}^{q} C_l^{**} x_i^{**(l)} + C_I \, t_{i,j} x_i^{*(2)}, \qquad j = 1, \ldots, n_i. \tag{17}$$

The intercept was $C_0 = 1.5$ and variables $x_i^{*(k)}$ for $k = 1, \ldots, 4$ have main effects with coefficient parameters $C_1^* = 2.5$, $C_2^* = 0$, $C_3^* = -1.2$, and $C_4^* = -0.2$. Variable $x_i^{*(2)}$ whose coefficient parameter is $C_2^* = 0$ has a linear interaction with time with coefficient parameter $C_I = -0.65$. Variables $x_i^{**(l)}$ for $l = 1, \ldots, q$ have coefficient parameters $C_l^{**} = 0$ and therefore are unrelated to $\mu_{i,j}$. Variables $x_i^{*(2)}$ and $x_i^{*(3)}$ were simulated from a uniform distribution on [1, 2] and [2, 3], respectively. All other variables were drawn from a standard normal distribution; all variables were drawn independently of one another. For each subject i, time values $t_{i,j}$ for $j = 1, \ldots, n_i$ were sampled with replacement from $\{1/N_0, 2/N_0, \ldots, 3\}$ where the number of time points n_i was drawn randomly from $\{1, \ldots, 3N_0\}$. This creates an unbalanced time structure because n_i is uniformly distributed over 1 to $3N_0$.

Simulation II The second simulation assumed the model:

$$\mu_{i,j} = C_0 + \sum_{k=1}^{4} C_k^* x_i^{*(k)} + \sum_{l=1}^{q} C_l^{**} x_i^{**(l)} + C_I \, t_{i,j}^2 \left(x_i^{*(2)}\right)^2. \tag{18}$$

This is identical to (17) except the linear feature-time interaction is replaced with a quadratic time trend and a quadratic effect in $x_i^{*(2)}$.

Simulation III The third simulation assumed the model:

$$\begin{aligned} \mu_{i,j} = {} & C_0 + C_1^* x_i^{*(1)} + C_3^* x_i^{*(3)} + C_4^* \exp(x_i^{*(4)}) \\ & + \sum_{l=1}^{q} C_l^{**} x_i^{**(l)} + C_I \, t_{i,j}^2 \left(x_i^{*(2)}\right)^2 x_i^{*(3)}. \end{aligned} \tag{19}$$

Model (19) is identical to (18) except variable $x_i^{*(4)}$ has a non-linear main effect and the feature-time interaction additionally includes $x_i^{*(3)}$.

5.1 Experimental settings

Four different experimental settings were considered, each with $n = \{100, 500\}$:

(A) $N_0 = 5$, and $q = 0$. For each i, $\mathbf{V}_i = \phi \mathbf{R}_i$ where $\phi = 1$ and $\mathbf{R}_i$ was an exchangeable correlation matrix with correlation $\rho = 0.8$ (i.e., $\text{Cov}(Y_{i,j}, Y_{i,k}) = \rho = 0.8$).
(B) Same as (A) except $N_0 = 15$.
(C) Same as (A) except $q = 30$.
(D) Same as (A) except $\text{Cov}(Y_{i,j}, Y_{i,j+k}) = \rho^k$ for $k = 0, 1, \ldots$ (i.e., AR(1) model).

5.2 Implementing boostmtree

All boostmtree calculations were implemented using the `boostmtree` R-package (Ishwaran et al. 2016), which implements the general boostmtree algorithm, Algorithm 3. The `boostmtree` package relies on the `randomForestSRC` R-package (Ishwaran and Kogalur 2016) for fitting multivariate regression trees. The latter is a generalization of univariate CART (Breiman et al. 1984) to the multivariate response setting and uses a normalized mean-squared error split-statistic, averaged over the responses, for tree splitting (see Ishwaran and Kogalur 2016, for details). All calculations used adaptive penalization cycling between (12) and (13). An exchangeable working correlation matrix was used where ρ was estimated using the in sample CV values $\{\widetilde{\boldsymbol{\mu}}_i^{(i,m)}\}_1^n$. All fits used cubic B-splines with 10 equally spaced knots subject to an $s = 3$ penalized differencing operator. Multivariate trees were grown to $K = 5$ terminal nodes. Boosting tuning parameters were set to $\nu = 0.05$ and $M = 500$ with the optimal number of boosting steps M_{opt} estimated using the in sample CV procedure.

5.3 Comparison procedures

5.3.1 GLS procedure

As a benchmark, we fit the data using a linear model under GLS that included all main effects for parameters and all pairwise linear interactions between **x**-variables and time. A correctly specified working correlation matrix was used. This method is called dgm-linear (dgm is short for data generating model).

5.3.2 Boosting comparison procedure

As a boosting comparison procedure we used the R-package `mboost` (Hothorn et al. 2010, 2016). We fit three different random intercept models. The first model was defined as

$$\text{mboost}_{\text{tr+bs}} \leftarrow \alpha_i \mathbf{1}_i + \text{btree}_K(\mathbf{x}_i) + \sum_{l=1}^{d} \text{btree}_K(\mathbf{x}_i, b_l(\mathbf{t}_i)).$$

The random intercept is denoted by α_i. The notation btree_K denotes a K-terminal node tree base learner. The first tree base learner is constructed using only the **x**-features, while the remaining tree-based learners are constructed using both **x**-features and time. The variable $b_l(\mathbf{t}_i)$ is identical to the lth B-spline time-basis used in boostmtree. The second model was

$$\text{mboost}_{\text{tr}} \leftarrow \alpha_i \mathbf{1}_i + \text{btree}_K(\mathbf{x}_i) + \text{btree}_K(\mathbf{x}_i, \mathbf{t}_i).$$

This is identical to the first model except time is no longer broken into B-spline basis terms. Finally, the third model was

$$\text{mboost}_{\text{bs}} \leftarrow \alpha_i \mathbf{1}_i + \text{btree}_K(\mathbf{x}_i) + \sum_{k=1}^{p} \text{bbs}_d(\mathbf{x}_i^{(k)} \star \mathbf{t}_i).$$

The term $\text{bbs}_d(\mathbf{x}_i^{(k)} \star \mathbf{t}_i)$ denotes all pairwise interactions between the kth **x**-variable and B-splines of order d. Thus, the third model incorporates all pairwise feature-time interactions. Notice that the first two terms in all three models are the same and therefore the difference in models depends on the base learner used for the third term. All three models were fit using

Table 1 Test set performance using simulations

	Experiment I				Experiment II				Experiment III			
	(A)	(B)	(C)	(D)	(A)	(B)	(C)	(D)	(A)	(B)	(C)	(D)
n = 100												
dgm-linear	**.356**	**.351**	**.421**	**.348**	.288	.294	.337	.287	.348	.356	.425	.349
$\text{mboost}_{\text{tr+bs}}$	.456	.445	.487	.438	.270	.256	.304	.269	.220	.198	.269	.221
$\text{mboost}_{\text{tr}}$	.449	.441	.478	.429	.253	.250	.277	.250	.192	.186	**.226**	.195
$\text{mboost}_{\text{bs}}$	.458	.451	.489	.434	**.233**	**.235**	**.250**	**.225**	.208	.209	.237	.205
boostmtree	.427	.417	.532	.414	**.237**	**.236**	.288	.231	**.173**	**.158**	**.226**	**.178**
boostmtree(.8)	.428	.416	.532	.423	.239	**.236**	.306	.240	.180	**.159**	.257	.190
n = 500												
dgm-linear	**.345**	**.344**	**.353**	**.342**	.283	.292	.289	.283	.344	.347	.355	.343
$\text{mboost}_{\text{tr+bs}}$	.399	.396	.394	.385	.216	.218	.219	.211	.135	.131	.149	.134
$\text{mboost}_{\text{tr}}$	.397	.395	.391	.384	.211	.217	.211	.206	.128	.128	.137	.126
$\text{mboost}_{\text{bs}}$	.398	.396	.394	.384	.206	.211	**.203**	.198	.175	.175	.176	.173
boostmtree	.368	.367	.392	.360	**.200**	**.208**	.214	**.193**	**.117**	**.115**	**.129**	**.114**
boostmtree(.8)	.368	.368	.390	.363	.202	**.210**	.219	.198	**.118**	**.115**	**.130**	**.115**

Values reported are test set standardized RMSE (sRMSE) averaged over 100 independent replications. Values displayed in bold identify the winning method for an experiment and any method within one standard error of its sRMSE

`mboost`. The number of boosting iterations was set to $M = 500$, however in order to avoid overfitting we use tenfold CV to estimate M_{opt}. All tree-based learners were grown to $K = 5$ terminal nodes. For all other parameters, we use default settings.

5.3.3 Other procedures

Several other procedures were used for comparison. However, because none compared favorably to boostmtree, we do not report these values here. For convenience some of these results are reported in Appendix 1.

5.4 RMSE performance

Performance was assessed using standardized root mean-squared error (sRMSE),

$$\text{sRMSE} = \frac{\left[\frac{1}{n}\sum_{i=1}^{n}\frac{1}{n_i}\sum_{j=1}^{n_i}(Y_{i,j} - \hat{Y}_{i,j})^2\right]^{1/2}}{\hat{\sigma}_Y}, \tag{20}$$

where $\hat{\sigma}_Y$ is the overall standard deviation of the response. Values for sRMSE were estimated using an independently drawn test set of size $n' = 500$. Each simulation was repeated 100 times independently and the average sRMSE value recorded in Table 1. Note that Table 1 includes the additional entry boostmtree(.8), which is boostmtree fit with ρ set at the specified value $\rho = 0.8$ (this yields a correctly specified correlation matrix for (A), (B), and (C)). Table 2 provides the standard error of the sRMSE values. Our conclusions are summarized below.

Table 2 Standard errors for Table 1 (multiplied by 1000)

	Experiment I				Experiment II				Experiment III			
	(A)	(B)	(C)	(D)	(A)	(B)	(C)	(D)	(A)	(B)	(C)	(D)
n = 100												
dgm-linear	2.93	2.94	3.72	2.22	1.33	1.25	1.99	1.42	2.72	2.13	2.54	2.32
$\text{mboost}_{\text{tr+bs}}$	4.46	4.38	4.77	3.19	1.58	1.51	2.57	2.18	2.48	1.71	3.16	2.63
$\text{mboost}_{\text{tr}}$	4.37	4.48	4.74	3.16	1.51	1.41	2.49	1.86	2.00	1.72	2.29	1.92
$\text{mboost}_{\text{bs}}$	4.52	4.54	4.94	3.01	1.52	1.55	1.98	1.71	2.25	1.69	2.17	2.03
boostmtree	5.19	4.48	7.24	4.12	1.89	1.69	3.51	1.98	2.73	1.79	3.38	3.44
boostmtree(.8)	5.11	4.39	7.13	4.14	1.95	1.65	3.36	2.35	2.90	1.85	5.06	5.44
n = 500												
dgm-linear	1.34	1.36	1.33	1.37	0.97	0.76	0.81	0.82	1.77	1.24	1.49	1.51
$\text{mboost}_{\text{tr+bs}}$	1.64	1.74	1.60	1.71	0.87	0.83	0.80	0.64	0.66	0.56	0.81	0.69
$\text{mboost}_{\text{tr}}$	1.63	1.74	1.55	1.71	0.82	0.80	0.82	0.60	0.55	0.53	0.66	0.55
$\text{mboost}_{\text{bs}}$	1.70	1.68	1.69	1.75	0.87	0.81	0.73	0.57	0.87	0.70	0.88	0.80
boostmtree	1.77	1.56	1.85	1.78	0.83	0.91	0.99	0.57	0.69	0.56	1.03	0.56
boostmtree(.8)	1.78	1.51	1.83	1.87	0.84	0.88	1.06	0.61	0.92	0.56	0.98	0.59

5.4.1 Experiment I

Performance of dgm-linear (the GLS model) is better than all other procedures in experiment I. This is not surprising given that dgm-linear is correctly specified in experiment I. Nevertheless, we feel performance of boostmtree is good given that it uses a large number of basis functions in this simple linear model with a single linear feature-time interaction.

5.4.2 Experiment II

In experiment II, $\text{mboost}_{\text{bs}}$, which includes all pairwise feature-time interactions, is correctly specified. However, interestingly, this seems only to confer an advantage over boostmtree for the smaller sample size $n = 100$. With a larger sample size ($n = 500$), performance of boostmtree is generally much better than $\text{mboost}_{\text{bs}}$.

5.4.3 Experiment III

Experiment III is significantly more difficult than experiments I and II since it includes a non-linear main effect as well as complex feature-time interaction. In this more complex experiment, boostmtree is significantly better than all mboost models, including $\text{mboost}_{\text{bs}}$, which is now misspecified.

5.4.4 Effect of correlation

In terms of correlation, the boostmtree procedure with estimated ρ is generally as good and sometimes even better than boostmtree using the correctly specified $\rho = 0.8$. Furthermore, loss of efficiency does not appear to be a problem when the working correlation matrix is misspecified as in simulation (D). In that simulation, the true correlation follows an AR(1)

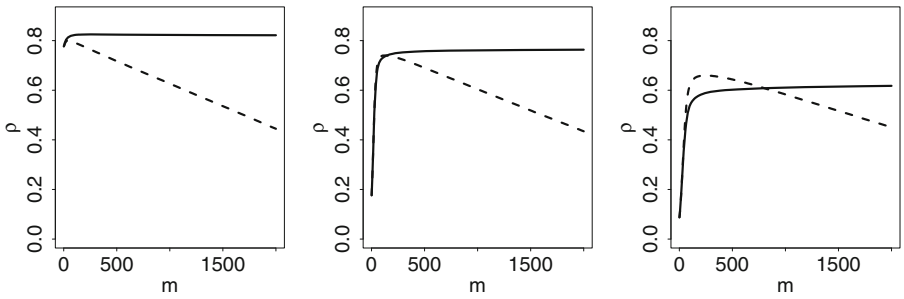

Fig. 1 Estimated correlation obtained using in sample CV (*solid line*) and without in sample CV (*dashed line*) for simulation experiment I (*left*), II (*middle*), and III (*right*)

model, yet performance of boostmtree under an exchangeable model is better for Experiment I and II, whereas results are comparable for Experiment III (compare columns (D) to columns (A)). We conclude that boostmtree using an estimated working correlation matrix exhibits good robustness to correlation.

5.5 In sample CV removes the rebound effect

In Sect. 4.4, we provided a theoretical explanation of the rebound effect for the correlation, and described how this could be corrected using the in sample CV predictor. In this Section, we provide empirical evidence demonstrating the effectiveness of this correction. For illustration, we used the 3 simulation experiments under experimental setting (A) with $n = 100$. The same boosting settings were used as before, except that we set $M = 2000$ and estimated ρ from (11) with and without the in sample CV method. Simulations were repeated 100 times independently. The average estimate of ρ is plotted against the boosting iteration m in Fig. 1.

As described earlier, among the 3 experiments, experiment I is the simplest, and experiment III is the most difficult. In all 3 experiments, the true value of ρ is 0.8. In experiment I, the estimate of ρ obtained using $\widetilde{\boldsymbol{\mu}}_i^{(i,m)}$ quickly reaches the true value, and remains close to this value throughout the entire boosting procedure, whereas the estimate of ρ obtained using $\boldsymbol{\mu}_i^{(m)}$ reaches the true value, but then starts to decline. This shows that the in sample CV method is able to eliminate the rebound effect. The rebound effect is also eliminated in experiments II and III using in sample CV, although now the estimated ρ does not reach the true value. This is less a problem in experiment II than III. This shows that estimating ρ becomes more difficult when the underlying model becomes more complex.

5.6 Accuracy of the in sample CV method

In this Section, we study the bias incurred in estimating M_{opt} and in estimating prediction error using the in sample CV method. Once again, we use the 3 simulation experiments under experimental setting (A). In order to study bias as a function of n, we use $n = \{100, 300, 500\}$. The specifications for implementing boostmtree are the same as before, but with $M = 2000$. The results are repeated using 100 independent datasets and 100 independent test data sets of size $n' = 500$. The results for M_{opt} are provided in Fig. 2. What we find are that the in sample CV estimates of M_{opt} are biased towards larger values, however bias shrinks towards zero with increasing n. We also observe that the in sample CV estimate is doing particularly well in experiment III.

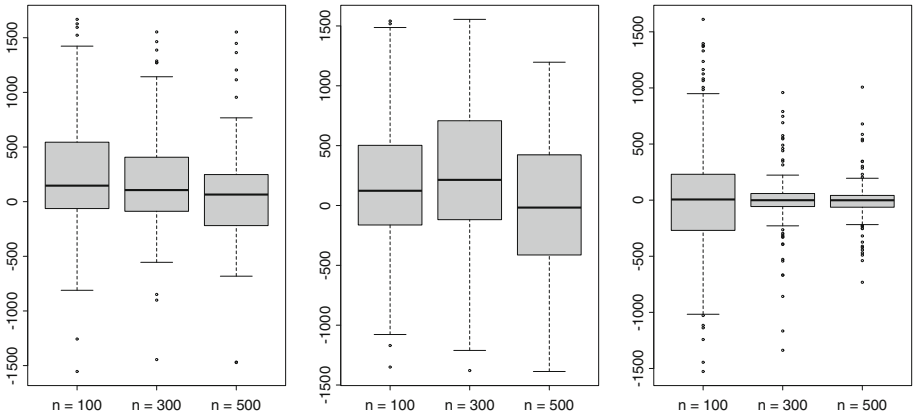

Fig. 2 Difference in the estimate of M_{opt} obtained using in sample CV to that obtained using test set data as a function of n. The *left*, *middle* and *right* plots are experiments I, II and III, respectively. In each case, we use 100 independent replicates

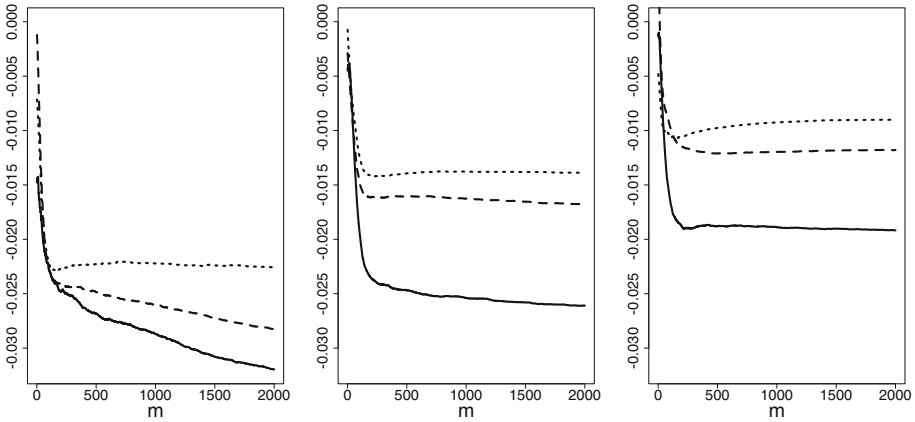

Fig. 3 Difference in the estimate of sRMSE obtained using in sample CV to that obtained using test set data. The *solid line* corresponds to $n = 100$, the *dashed line* corresponds to $n = 300$, and the *dotted line* corresponds to $n = 500$. The *left*, *middle* and *right* plots are experiments I, II and III, respectively. Values are averaged over 100 independent replicates

Results summarizing the accuracy in estimating prediction error are provided in Fig. 3. The vertical axis displays the difference in standardized RMSE estimated using $\widetilde{\text{CV}}^{(m)}/\hat{\sigma}_Y$ from the in sample CV method and using (20) by direct test set calculation. This shows an optimistic bias effect for the in sample CV method, which is to be expected, however bias is relatively small and diminishes rapidly as n increases. To better visualize the size of this bias, consider Fig. 4 ($n = 500$ for all three experiments). This shows that in sample CV estimates are generally close to those obtained using a true test set.

5.7 Feature selection

We used permutation variable importance (VIMP) for feature selection. In this method, let $\mathbf{X} = \left[\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(p)}\right]_{n' \times p}$ represent the test data where $\mathbf{x}_{(k)} = (x_{1,k}, \ldots, x_{n',k})^T$ records all

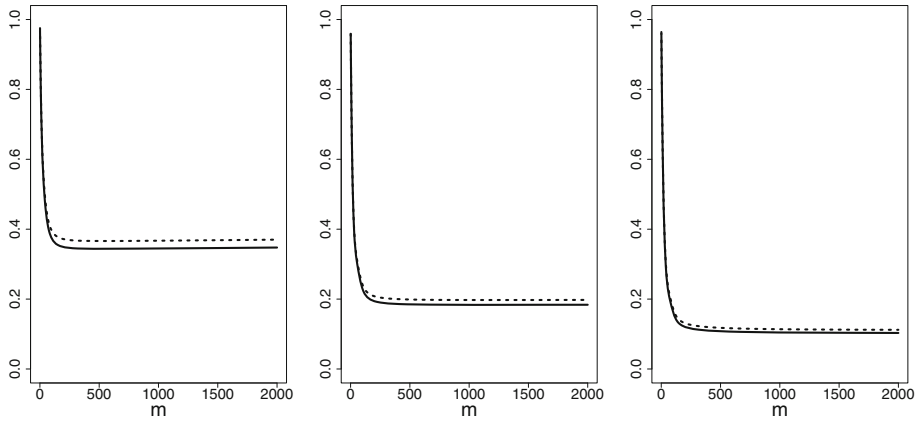

Fig. 4 Estimated sRMSE obtained using in sample CV (*solid line*) and obtained using test set data (*dotted line*) for $n = 500$. The *left*, *middle* and *right* plots are experiments I, II and III, respectively. Values are averaged over 100 independent replicates

test set values for the kth feature, $k = 1, 2, \ldots, p$. At each iteration $m = 1, \ldots, M$, the test data $\mathbf{X}$ is run down the mth tree (grown previously using training data). The resulting node membership is used to determine the estimate of $\boldsymbol{\beta}$ for the mth iteration, denoted by $\hat{\boldsymbol{\beta}}^{(m)}$. Let $\mathbf{x}^*_{(k)} = \left(x_{j_1,k}, \ldots, x_{j_{n'},k}\right)^T$ represent the kth feature after being "noised-up" by randomly permuting the coordinates of the original $\mathbf{x}_{(k)}$. Using $\mathbf{x}^*_{(k)}$, a new test data $\mathbf{X}_k = [\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(k-1)}, \mathbf{x}^*_{(k)}, \mathbf{x}_{(k+1)}, \ldots, \mathbf{x}_{(p)}]$ is formed by replacing $\mathbf{x}_{(k)}$ with the noised up $\mathbf{x}^*_{(k)}$. The new test data $\mathbf{X}_k$ is run down the mth tree and from the resulting node membership used to estimate $\boldsymbol{\beta}$, which we call $\hat{\boldsymbol{\beta}}^{(m)}_k$. The first coordinate of $\hat{\boldsymbol{\beta}}^{(m)}_k$ reflects the contribution of noising up the main effect $\beta_0(\mathbf{x})$, while the remaining d coordinates reflect noising up the feature-time interactions $\sum_{l=1}^{d} \mathbf{b}_l(\mathbf{t})\beta_l(\mathbf{x})$. Comparing the performance of the predictor obtained using $\hat{\boldsymbol{\beta}}^{(m)}_k$ to that obtained using the non-noised up $\hat{\boldsymbol{\beta}}^{(m)}$ yields an estimate of the overall importance of the feature k.

However, in order to isolate whether feature k is influential for the main effect alone, removing any potential effect on time it might have, we define a modified noised up estimator $\hat{\boldsymbol{\beta}}^{(m)}_{k,1}$ as follows. The first coordinate of $\hat{\boldsymbol{\beta}}^{(m)}_{k,1}$ is set to the first coordinate of $\hat{\boldsymbol{\beta}}^{(m)}_k$, while the remaining d coordinates are set to the corresponding coordinates of $\hat{\boldsymbol{\beta}}^{(m)}$. By doing so, any effect that $\hat{\boldsymbol{\beta}}^{(m)}_{k,1}$ may have is isolated to a main effect only. Denote the test set predictor obtained from $\hat{\boldsymbol{\beta}}^{(m)}_{k,1}$ and $\hat{\boldsymbol{\beta}}^{(m)}$ by $\hat{\boldsymbol{\mu}}^{(m)}_{k,1}$ and $\hat{\boldsymbol{\mu}}^{(m)}$. The difference between the test set RMSE for $\hat{\boldsymbol{\mu}}^{(m)}_{k,1}$ and $\hat{\boldsymbol{\mu}}^{(m)}$ is defined as the VIMP main effect for feature k.

In a likewise fashion, a noised up estimator $\hat{\boldsymbol{\beta}}^{(m)}_{k,2}$ measuring noising up for feature-time interactions (but not main effects) is defined analogously. The first coordinate of $\hat{\boldsymbol{\beta}}^{(m)}_{k,2}$ is set to the first coordinate of $\hat{\boldsymbol{\beta}}^{(m)}$ and the remaining d coordinates to the corresponding values of $\hat{\boldsymbol{\beta}}^{(m)}_k$. The difference between test set RMSE for $\hat{\boldsymbol{\mu}}^{(m)}_{k,2}$ and $\hat{\boldsymbol{\mu}}^{(m)}$ equals VIMP for the feature-time effect for feature k. Finally, to assess an overall effect of time, we randomly permute the rows of the matrix $\{\mathbf{D}_i\}_1^n$. The resulting predictor $\hat{\boldsymbol{\mu}}^{(m)}_t$ is compared with $\hat{\boldsymbol{\mu}}^{(m)}$ to determine an overall VIMP time effect.

To assess boostmtree's ability to select variables we re-ran our previous experiments under setting (C) with $n = 100$ and $q = 10, 25, 100$. Recall q denotes the number of non-outcome related variables (i.e. zero signal variables). Thus increasing q increases dimension but keeps

Table 3 Standardized VIMP averaged over 100 independent replications for variables $x^{*(1)}, \ldots, x^{*(4)}$, non-outcome related variables $\{x^{**(l)}\}_1^q$ (values averaged over $l = 1, \ldots, q$ and denoted by noise), and time. VIMP is separated into main effects of feature, time, and feature-time interaction effects

EXPT	VIMP effect for features					VIMP interaction effect for features and time					VIMP effect for time
	1	2	3	4	Noise	1	2	3	4	Noise	
No of noise variables $q = 10$											
I	107	−0.3	10	0.4	−0.3	0.4	3	0.2	0.1	0	29
II	91	0.3	8	0.3	0	1	90	0.6	0	0.1	312
III	44	5	16	0.7	0.1	−0.3	136	97	−0.1	0	446
No of noise variables $q = 25$											
I	94	−0.1	7	0.4	−0.1	0.2	2	0.2	0.1	0	25
II	80	0.8	5	0.4	0	0.7	82	0.1	0.2	0.1	284
III	38	5	11	0.6	0.1	2	120	83	0	0	399
No of noise variables $q = 100$											
I	53	−0.2	2	0	0	0	0.9	0	0	0	16
II	42	2	1	0.1	0	0.8	54	0	0	0	202
III	16	3	11	0.1	0	0.1	76	51	0	0	288

signal strength fixed. We divided all VIMP values by the RMSE for $\hat{\boldsymbol{\mu}}^{(m)}$ and then multiplied by 100. We refer to this as standardized VIMP. This value estimates importance relative to the model: large positive values identify important effects. Standardized VIMP was recorded for each simulation. Simulations were repeated 100 times independently and VIMP values averaged.

Table 3 records standardized VIMP for main effects and feature-time effects for variables $x^{*(1)}, \ldots, x^{*(4)}$. Standardized VIMP for non-outcome related variables $\{x^{**(l)}\}_1^q$ were averaged and appear under the column entry "noise". Table 3 shows that VIMP for noise variables are near zero, even for $q = 100$. VIMP for signal variables in contrast are generally positive. Although VIMP for $x^{*(4)}$ is relatively small, especially in high-dimension $q = 100$, this is not unexpected as the variable contributes very little signal. Delineation of main effect and time-interactions is excellent. Main effects for $x^{*(1)}$ and $x^{*(3)}$ are generally well identified. The feature-time interaction of $x^{*(2)}$ is correctly identified in experiments II and III, which is impressive given that $x^{*(2)}$ has a time-interaction but no main effect. The interaction is not as well identified in experiment I. This is because in experiment I, the interaction is linear and less discernible than experiments II and III, where the effect is quadratic. Finally, the time-interaction of $x^{*(3)}$ in experiment III is readily identified even when $q = 100$.

6 Postoperative spirometry after lung transplantation

Forced 1-second expiratory volume (FEV1) is an important clinical outcome used to monitor health of patients after lung transplantation (LTX). FEV1 is known (and expected) to decline after transplantation, with rate depending strongly on patient characteristics; however, the relationship of FEV1 to patient variables is not fully understood. In particular, the benefit of double versus single lung transplant (DLTX versus SLTX) is debated, particularly because pulmonary function is only slightly better after DLTX.

Table 4 Variable names from spirometry analysis

Height	Height of patient
Weight	Weight of patient
FEVPN_PR	Forced expiratory volume in 1 s, normalized, Pre-transplantation
Age	Age at transplant
Female	Female patient
BSA	Body surface area
BMI	Body Mass Index
RaceW	White race
RaceB	Black race
ABO variables	Blood types A, B, AB, and O
TRACH_PR	Pre-transplant tracheostomy
EISE	Eisenmenger disease
PPH	Primary pulmonary hypertension
IPF	Idiopathic pulmonary fibrosis
SARC	Sarcoidosis
ALPH	Alpha-antitrypsin disease
COPD	Chronic obstructive pulmonary disease
DLTX	Double lung transplantation
Left	Left lung transplant
Right	Right lung transplant

Using FEV1 longitudinal data collected at the Cleveland Clinic (Mason et al. 2012), we sought to determine clinical features predictive of FEV1 and to explore the effect of DLTX and SLTX on FEV1 allowing for potential time interactions with patient characteristics. In total, 9471 FEV1 evaluations were available from 509 patients who underwent lung transplantation from the period 1990 through 2008 (median follow up for all patients was 2.30 years). On average, there were over 18 FEV1 measurements per patient; 46 % of patients received two lungs, and for patients receiving single lungs, 49 % (nearly half) received left lungs. In addition to LTX surgery status, 18 additional patient clinical variables were available. Table 4 provides definitions of the variables used in the analysis. Table 5 describes summary statistics for patients, stratified by lung transplant status.

As before, calculations were implemented using the `boostmtree` R-package. An exchangeable working correlation matrix was used for the boostmtree analysis. Adaptive penalization was applied using cubic B-splines with 15 equally spaced knots under a differencing penalization operator of order $s = 3$. Number of boosting iterations was set to $M = 1000$ with in sample CV used to determine M_{opt}. Multivariate trees were grown to a depth of $K = 5$ terminal nodes and $\nu = .01$. Other parameter settings were informally investigated but without noticeable difference in results. The data was randomly split into training and testing sets using an 80/20 split. The test data set was used to calculate VIMP.

Figure 5 displays predicted FEV1 values against time, stratified by LTX status (for comparison, see Appendix 2 for predicted values obtained using the mboost procedures considered in Sect. 5). Double lung recipients not only have higher FEV1 but values decline more slowly, thus demonstrating an advantage of the increased pulmonary reserve provided by double lung transplant. Figure 6 displays the standardized VIMP for main effects and feature-time interactions for all variables. The largest effect is seen for LTX surgery status, which accounts for

Table 5 Summary statistics of patient variables for spirometry data

	All patients (n = 509)	Single transplant (n = 245)	Double transplant (n = 264)
Age	49.34 ± 12.90	57.22 ± 7.05	42.03 ± 12.80
Sex (F)	242 (48)	110 (45)	132 (50)
Height	167.78 ± 10.13	168.10 ± 9.75	167.47 ± 10.49
Weight	68.86 ± 17.23	70.77 ± 15.25	67.10 ± 18.73
BMI	24.33 ± 5.23	24.97 ± 4.63	23.75 ± 5.68
BSA	1.80 ± 0.27	1.83 ± 0.24	1.77 ± 0.29
FEVPN_PR	28.54 ± 15.38	27.21 ± 14.18	29.78 ± 16.34
RaceW	472 (93)	236 (96)	236 (89)
Blood Gr(A)	210 (41)	103 (42)	107 (41)
Blood Gr(AB)	18 (4)	9 (4)	9 (3)
Blood Gr(B)	61 (12)	22 (9)	39 (15)
TRACH_PR	1 (0)	0 (0)	1(0)
EISE	7 (1)	0 (0)	7 (3)
PPH	18 (4)	2 (1)	16 (6)
IPF	96 (9)	50 (20)	46 (17)
SARC	19 (4)	6 (2)	13 (5)
ALPH	34 (7)	23 (9)	11 (4)
COPD	202 (40)	148 (60)	54 (20)

Values in the table are mean ± standard deviation or $n(\%)$, where n denotes the sample size

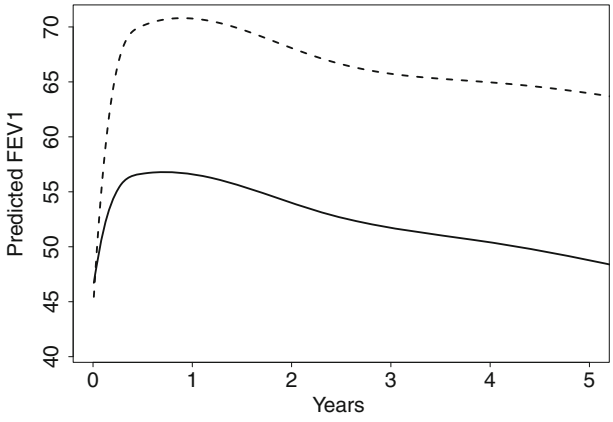

Fig. 5 Predicted FEV1 versus time stratified by single lung SLTX (*solid line*) and double lung DLTX (*dashed line*) status

nearly 10% of RMSE. Interestingly, this is predominately a time-interaction effect (that no main effect was found for LTX is corroborated by Fig. 5 which shows FEV1 to be similar at time zero between the two groups). In fact, many of the effects are time-interactions, including a medium sized effect for age. Only FEVPN_PR (pre-transplantation FEV1) appears to have a main effect, although the standardized VIMP is small.

The LTX and age time-interaction findings are interesting. In order to explore these relationships more closely we constructed partial plots of FEV1 versus age, stratified by LTX

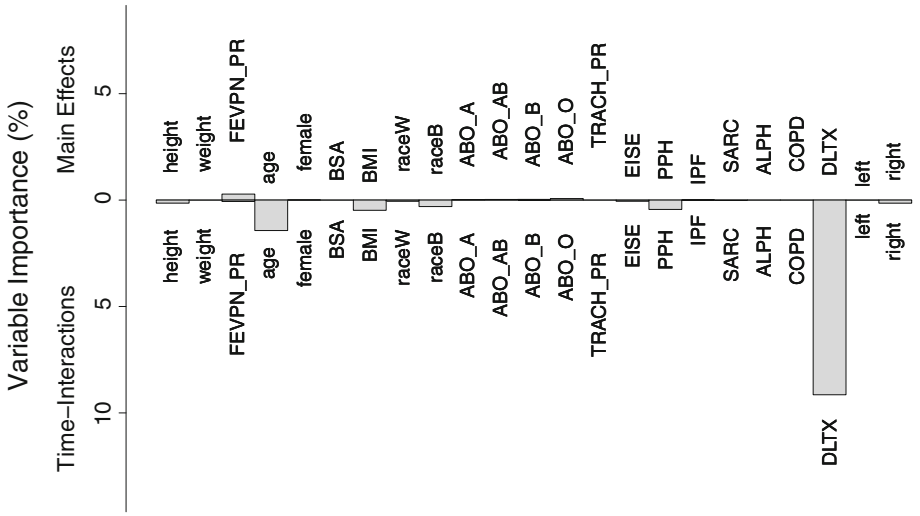

Fig. 6 Standardized variable importance (VIMP) for each feature from boostmtree analysis of spirometry longitudinal data. *Top* values are main effects only; *bottom* values are time-feature interaction effects

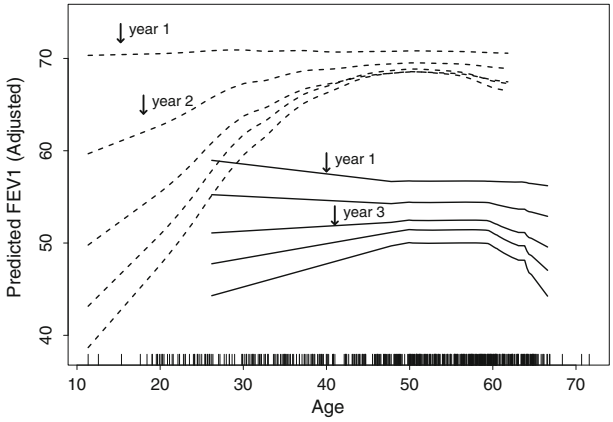

Fig. 7 Partial plot of FEV1 versus age stratified by single lung SLTX (*solid lines*) and double lung DLTX (*dashed lines*) treatment status evaluated at years 1, . . . , 5

(Fig. 7). The vertical axis displays the adjusted partial predicted value of FEV1, adjusted for all features (Friedman 2001). The relationship between FEV1 and age is highly dependent on LTX status. DLTX patients have FEV1 responses which increase rapidly with age, until about age 50 where the curves flatten out. Another striking feature is the time dependency of curves. For DLTX, increase in FEV1 in age becomes sharper with increasing time, whereas for SLTX, although an increase is also seen, it is far more muted.

The general increase in FEV1 with age is interesting. FEV1 is a measure of a patient's ability to forcefully breathe out and in healthy patients FEV1 is expected to decrease with age. The explanation for the reverse effect seen here is due to the state of health of lung transplant patients. In our cohort, older patients tend to be healthier than younger patients, who largely suffer from incurable diseases such as cystic fibrosis, and who therefore produce smaller FEV1 values. This latter group is also more likely to receive double lungs. Indeed,

they likely make up the bulk of the young population in DLTX. This is interesting because not only does it explain the reverse effect, but it also helps explain the rapid decrease in FEV1 observed over time for younger DLTX patients. It could be that over time the transplanted lung is reacquiring the problems of the diseases in this subgroup. This finding appears new and warrants further investigation in the literature.

7 Discussion

Trees are computationally efficient, robust, model free, highly adaptive procedures, and as such are ideal base learners for boosting. While boosted trees have been used in a variety of settings, a comprehensive framework for boosting multivariate trees in longitudinal data settings has not been attempted. In this manuscript we described a novel multivariate tree boosting method for fitting a semi-nonparametric marginal model. The boostmtree algorithm utilizes P-splines with estimated smoothing parameter and has the novel feature that it enables nonparametric modeling of features while simultaneously smoothing semi-nonparametric feature-time interactions. Simulations demonstrated boostmtree's ability to estimate complex feature-time effects; its robustness to misspecification of correlation; and its effectiveness in high dimensions. The applicability of the method to real world problems was demonstrated using a longitudinal study of lung transplant patients. Without imposing model assumptions we were able to identify an important clinical interaction between age, transplant status, and time. Complex two-way feature-time interactions such as this are rarely found in practice and yet we were able to discover ours with minimal effort through our procedure.

All boostmtree calculations in this paper were implemented using the `boostmtree` R-package (Ishwaran et al. 2016) which is freely available on the Comprehensive R Archive Network (https://cran.r-project.org). The `boostmtree` package relies on the `randomForestSRC` R-package (Ishwaran and Kogalur 2016) for fitting multivariate regression trees. Various options are available within `randomForestSRC` for customizing the tree growing process. In the future, we plan to incorporate some of these into the `boostmtree` package. One example is non-deterministic splitting. It is well known that trees are biased towards favoring splits on continuous features and factors with a large numbers of categorical levels (Loh and Shih 1997). To mitigate this bias, `randomForestSRC` provides an option to select a maximum number of split-points used for splitting a node. The splitting rule is applied to the random split points and the node is split on that feature and random split point yielding the best value (as opposed to deterministic splitting where all possible split points are considered). This mitigates tree splitting bias and reduces bias in downstream inference such as feature selection. Other tree building procedures, also designed to mitigate feature selection bias (Hothorn et al. 2006), may also be incorporated in future versions of the `boostmtree` software. Another important extension to the model (and software) worthy of future research will be the ability to handle time-dependent features. In this paper we focused exclusively on time-independent features. One reason for proposing model (1) is that it is difficult to deal with multiple time-dependent features using tree-based learners. The problem of handling time-dependent features is a known difficult issue with binary trees due to the non-uniqueness in assigning node membership—addressing this remains an open problem for multivariate trees. None of this mitigates the usefulness of model (1), but merely points to important and exciting areas for future research.

Acknowledgments This work was supported by the National Institutes of Health (R01CA16373 to H.I. and U.B.K., RO1HL103552 to H.I., J.R., J.E., U.B.K. and E.H.B).

Appendix 1: Other comparison procedures

Section 5 used mboost as a comparison procedure to boostmtree. However because mboost does not utilize a smoothing parameter over feature-time interactions, it is reasonable to wonder how other boosting procedures using penalization would have performed. To study this, we consider likelihood boosting for generalized additive models using P-splines (Groll and Tutz 2012). For compuations we use the R-function `bGAMM` from the `GMMBoost` package. In order to evaluate performance of the `bGAMM` procedure, we consider the first experimental setting (A) for each of the three experiments in Sect. 5. For models, we used all features for main effects and P-splines for feature-time interactions. The `bGAMM` function requires specifying a smoothing parameter. This value is optimized by repeated fitting of the function over a grid of smoothing parameters and choosing that value minimizing AIC. We used a grid of smoothing parameters over [1, 1000] with increments of roughly 100 units. All experiments were repeated over 20 independent datasets (due to the length of time taken to apply `bGAMM` we used a smaller number of replicates than in Sect. 5). The results are recorded in Fig. 8. We find `bGAMM` does well in Experiment I as it is correctly specified here by involving only linear main effects and a linear feature-time interaction. But in Experiments II and III, which involve nonlinear terms and more complex interactions, performance of `bGAMM` is substantially worse than boostmtree (this is especially true for Experiment III which is the most complex model).

Next we consider RE-EM trees (Sela and Simonoff 2012), which apply to longitudinal and cluster unbalanced data and time varying features. Let $\{y_{i,j}, \mathbf{x}_{i,j}\}_1^{n_i}$ denote repeated measurements for subject i. RE-EM trees fit a normal random effects model, $y_{i,j} = \mathbf{z}_{i,j}^T \boldsymbol{\beta}_i + f(\mathbf{x}_{i,j}) + \varepsilon_{i,j}$, for $j = 1, 2, \ldots, n_i$, where $\mathbf{z}_{i,j}$ are features corresponding to the random effect $\boldsymbol{\beta}_i$. RE-EM uses a two-step fitting procedure. At each iteration, the method alternates between: (a) fitting a tree using the residual $y_{i,j} - \mathbf{z}_{i,j}^T \hat{\boldsymbol{\beta}}_i$ as the response and $\mathbf{x}_{i,j}$ as features; and (b) fitting a mixed effect model upon substituting the tree estimated value for $f(\mathbf{x}_{i,j})$. We compare test set performance of RE-EM trees to boostmtree using experimental setting (A) of Sect. 5. RE-EM trees was implemented using the R-package `REEMtree`. Figure 9 displays the results and shows clear superiority of boostmtree.

Appendix 2: Comparing predicted FEV1 using boostmtree and mboost

Section 6 presented an analysis of the spirometry data using boostmtree. Figure 5 plotted the predicted FEV1 against time (stratified by single/double lung transplant status), where the predicted value for FEV1 was obtained using boostmtree. In Fig. 10 below, we compare the

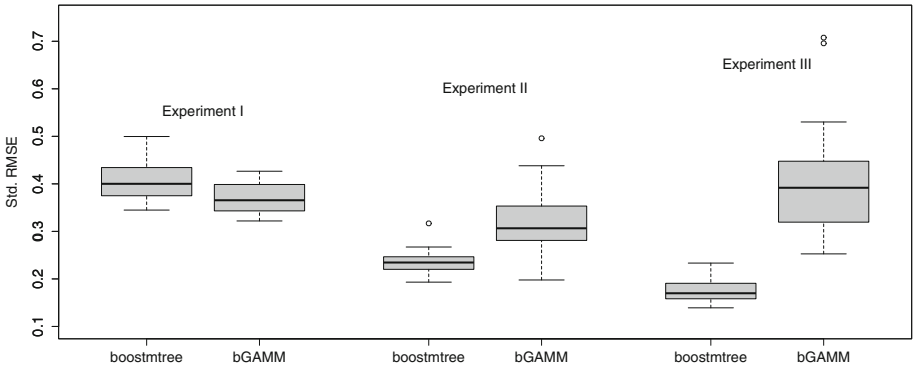

Fig. 8 Test set performance of `bGAMM` versus boostmtree using 20 independent datasets

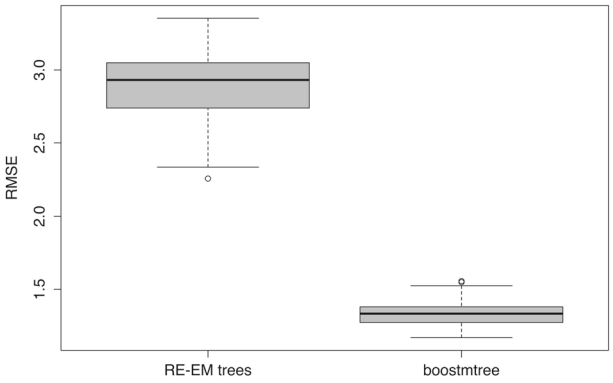

Fig. 9 Test set performance of RE-EM trees versus boostmtree using 100 independent datasets

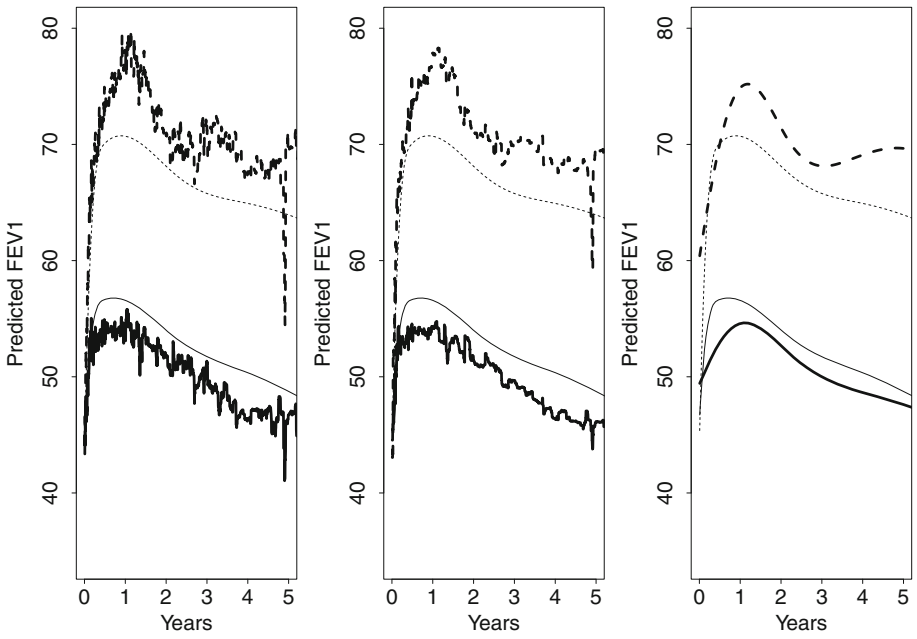

Fig. 10 Predicted FEV1 versus time stratified by single lung SLTX (*solid line*) and double lung DLTX (*dashed line*) status. *Thin lines* displayed in each of *three plots* are boostmtree predicted values. *Thick lines* are: $\text{mboost}_{\text{tr+bs}}$ (*left*), $\text{mboost}_{\text{tr}}$ (*middle*), and $\text{mboost}_{\text{bs}}$ (*right*)

boostmtree predicted FEV1 to the three mboost models considered earlier in Sect. 5. Settings for mboost were the same as considered in Sect. 5, with the exception that the total number of boosting iterations was set to $M = 1000$. Figure 10 shows that the overall trajectory of predicted FEV1 is similar among all procedures. However compared to boostmtree, mboost models underestimate predicted FEV1 for single lung transplant patients, and overestimate FEV1 for double lung transplant patients. It is also interesting that $\text{mboost}_{\text{tr+bs}}$ and $\text{mboost}_{\text{tr}}$ are substantially less smooth than $\text{mboost}_{\text{bs}}$.

References

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. California: Belmont.

De Boor, C. (1978). *A practical guide to splines*. Berlin: Springer.

Diggle, P., Heagerty, P., Liang, K.-Y., & Zeger, S. (2002). *Analysis of longitudinal data*. Oxford: Oxford University Press.

Duchon, J. (1977). Splines minimizing rotation-invariant semi-norms in Sobolev spaces. In *Constructive theory of functions of several variables* (pp. 85–100). Berlin Heidelberg: Springer.

Eilers, P. H. C., & Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. *Statistical Science, 11*(2), 89–102.

Freund, Y., & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the 13th international conference on machine learning* (pp. 148–156).

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*, 1189–1232.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, *38*(4), 367–378.

Groll, A., & Tutz, G. (2012). Regularization for generalized additive mixed models by likelihood-based boosting. *Methods of Information in Medicine*, *51*(2), 168.

Hastie, T. J., & Tibshirani, R. J. (1990). *Generalized additive models* (Vol. 43). Boca raton: CRC Press.

Hoover, D. R., Rice, J. A., Wu, C. O., & Yang, L.-P. (1998). Nonparametric smoothing estimates of time-varying coefficient models with longitudinal data. *Biometrika*, *85*(4), 809–822.

Hothorn, T., Hornik, K., & Zeileis, A. (2006). Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, *15*, 651–674.

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

Hothorn, T., Buhlmann, P., Kneib, T., Schmid, M., Hofner, B., Sobotka, A., & Scheipl, F. (2016). *mboost: Model-based boosting*, 2016. R package version 2.6-0.

Ishwaran, H., & Kogalur, U. B. (2016). *Random forests forsurvival, regression and classification (RF-SRC)*, 2016. R packageversion 2.2.0.

Ishwaran, H., Pande, A., & Kogalur, U. B. (2016). *Boostmtree: Boosted multivariate trees for longitudinaldata*, 2016. R package version 1.1.0.

Loh, W.-Y., & Shih, Y.-S. (1997). Split selection methods for classification trees. *Statistica Sinica*, *7*, 815–840.

Mallat, S., & Zhang, Z. (1993). Matching pursuits with time-frequency dictionaries. *IEEE Transactions on Signal Processing*, *41*, 3397–3415.

Mason, D. P., Rajeswaran, J., Liang, L., Murthy, S. C., Su, J. W., Pettersson, G. B., et al. (2012). Effect of changes in postoperative spirometry on survival after lung transplantation. *The Journal of Thoracic and Cardiovascular Surgery*, *144*(1), 197–203.

Mayr, A., Hothorn, T., & Fenske, N. (2012). Prediction intervals for future BMI values of individual children-a non-parametric approach by quantile boosting. *BMC Medical Research Methodology*, *12*(1), 6.

Mayr, A., Hofner, B., & Schmid, M. (2012). The importance of knowing when to stop: A sequential stopping rule for component-wise gradient boosting. *Methods of Information in Medicine*, *51*, 178–186.

Pan, W. (2001). Akaike's information criteria in generalized estimating equations. *Biometrika*, *57*, 120–125.

Pinheiro, J. C., & Bates, D. M. (2000). *Mixed-effects models in S and S-PLUS*. Berlin: Springer.

Pinheiro, J.C., Bates, D.M., DebRoy, S., Sarkar, D., & R Core Team. (2014).*nlme: Linear and nonlinear mixed effects models*. Rpackage version 3.1-117.

Robinson, G. K. (1991). That BLUP is a good thing: The estimation of random effects. *Statistical Science, 6*(1), 15–32.

Ruppert, D., Wand, M. P., & Carroll, R. J. (2003). *Semiparametric regression*. (Vol. 12). Cambridge: Cambridge University Press.

Sela, R. J., & Simonoff, J. S. (2012). RE-EM trees: A data mining approach for longitudinal and clustered data. *Machine Learning*, *86*, 169–207.

Tutz, G., & Binder, H. (2006). Generalized additive modeling with implicit variable selection by likelihood-based boosting. *Biometrics*, *62*(4), 961–971.

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

Wahba, G. (1990). *Spline models for observational data* (Vol. 59). Bangkok: SIAM.