manuscript 00-00

Boost-R: Gradient Boosted Trees for Recurrence Data

Xiao Liu
Department of Industrial Engineering, University of Arkansas
University of Arkansas, xl027@uark.edu

Rong Pan
School of Computing, Informatics, Decision Systems Engineering
Arizona State University, rong.pan@asu.edu

Recurrence data arise from multi-disciplinary domains spanning reliability, cyber security, healthcare, online retailing, etc. This paper investigates an additive-tree-based approach, known as Boost-R (Boosting for Recurrence Data), for recurrent event data with both static and dynamic features. Boost-R constructs an ensemble of gradient boosted additive trees to estimate the cumulative intensity function of the recurrent event process, where a new tree is added to the ensemble by minimizing the regularized L^2 distance between the observed and predicted cumulative intensity. Unlike conventional regression trees, a time-dependent function is constructed by Boost-R on each tree leaf. The sum of these functions, from multiple trees, yields the ensemble estimator of the cumulative intensity. The divide-and-conquer nature of tree-based methods is appealing when hidden sub-populations exist within a heterogeneous population. The non-parametric nature of regression trees helps to avoid parametric assumptions on the complex interactions between event processes and features. Critical insights and advantages of Boost-R are investigated through comprehensive numerical examples. Datasets and computer code of Boost-R are made available on GitHub. To our best knowledge, Boost-R is the first gradient boosted additive-tree-based approach for modeling large-scale recurrent event data with both static and dynamic feature information.

Key words: Additive Trees, Recurrent Event Data, Gradient Boosting, Reliability, Feature Selection

1. Introduction

1.1. Background

The rapid penetration of IoT technologies gives rise to large-scale recurrent event data from multidisciplinary domains, including reliability, asset management, clinical trials, cyber security, etc. A typical recurrent event dataset has two defining characteristics: (***i***) the event occurrence times experienced by individuals are recorded; (***ii***) each individual is characterized by both static and dynamic feature/covariate information. By learning the relationship between event processes and features, statistical approaches are needed to better understand why critical events happened in the past, when events of interest will recur in the future, and how one could optimize the event processes through proactive interventions.

For example, Liu and Pan (2020) considered an asset management problem of a large fleet of oil and gas wells. An individual well, throughout its production life, requires repeated maintenance due to failures over time. Because oil and gas wells are often located in vast and remote spatial areas, the capabilities of predicting failures greatly facilitate maintenance planning and reduce total production cost. For individual wells, times of failures (i.e., repairs) are well documented. As a motivating example, the left panel of Figure 1 shows the geo-locations of 8232 oil and gas wells installed between 2007 and 2017, while the right panel of Figure 1 shows the failure processes of a pump-related failure mode for 40 selected wells.

In this example, eight static well attributes are available, $x_1, x_2, ..., x_8$. In particular, $x_1, x_2, ..., x_6$ are the static well attributes, including the dimensions of two critical components (x_1 and x_2), average stroke length (x_3), mean-time-between-failure (x_4), mean polished rod horse power (x_5), and mean load on pump plunger against position (x_6). The last two attributes, x_7 and x_8, are the well latitude and longitude. Figure 2 shows the (standardized) values of $x_1, x_2, ..., x_6$ for all 8232 systems. It is seen that these systems have different static well attributes.

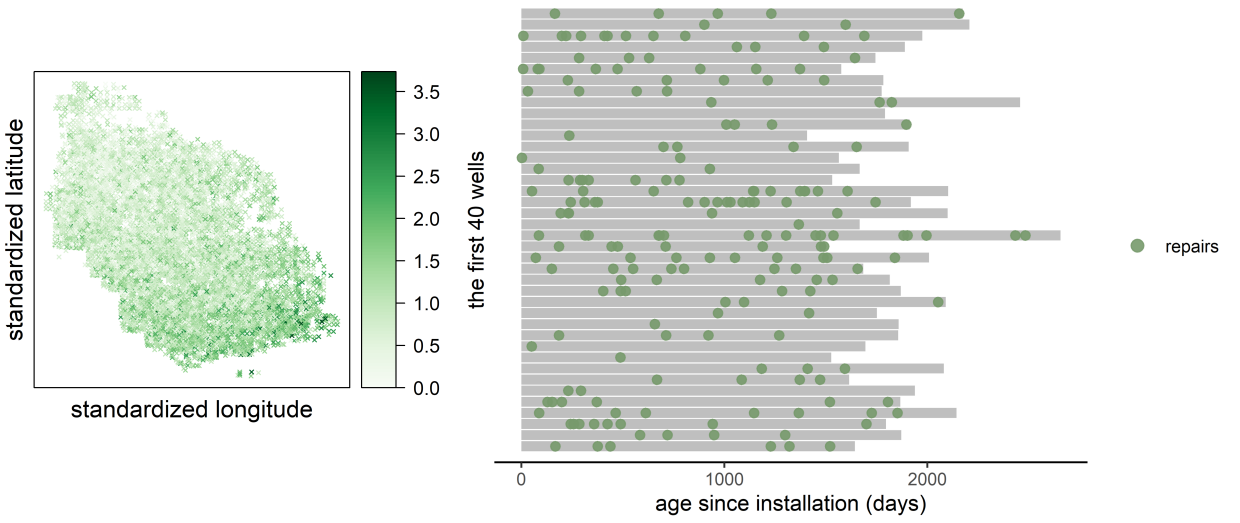

Figure 1 **Left panel: geo-locations of 8232 wells with the legend showing the average number of failures per year; Right panel: illustration of the failure processes of the first 40 wells.**

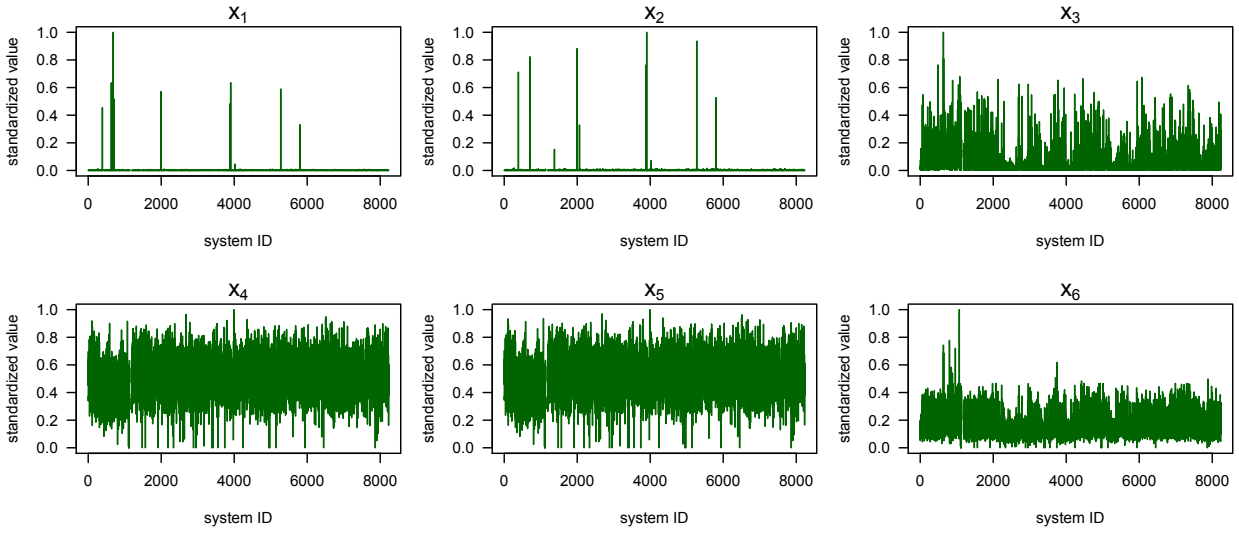

Figure 2 **Standardized values of $x_1, x_2, ..., x_6$ for all 8232 systems.**

In addition to the static well attributes, critical well operating conditions are monitored by sensors, including gear box torque, stroke length, polished rod horse power, peak surface load, pump card area and cycles. For illustrative purposes, Figure 3 shows the (standardized) gear torque for five chosen systems. It is seen from Figure 3 that these systems experience different operating conditions, which are rarely synchronized and lead to further heterogeneity in how wells fail over time.

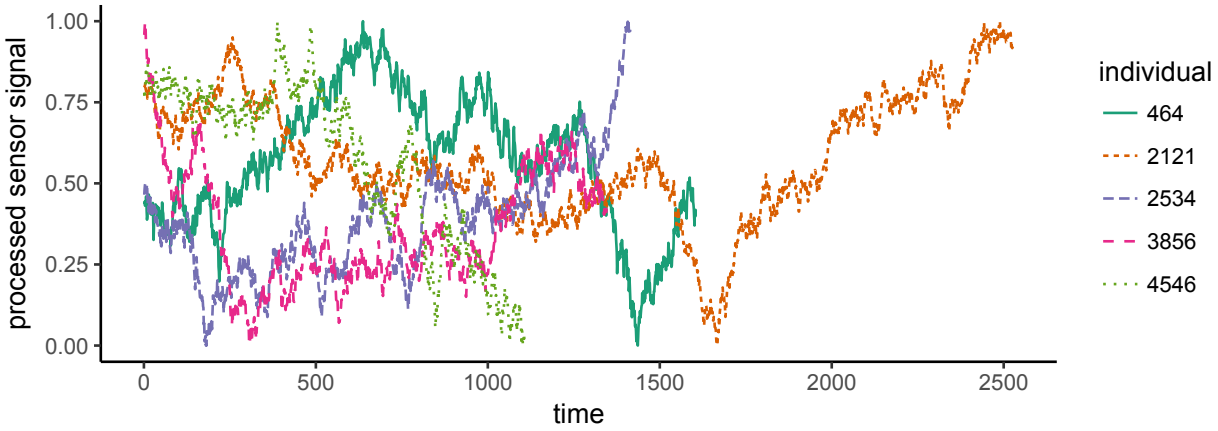

Figure 3 **Gear torque for 5 well systems**

More application examples of recurrent event data can be found from other application domains. In personalised healthcare, it is crucial to understand the interaction between the recurrence of a chronic medical condition and a patient's risk factor; In cyber security, of interest are often the prediction of recurrent fraudulent activities on susceptible systems within a dynamic cyber environment; In transportation safety, it is important to reveal the connection between the recurrences of accidents and the traffic/visibility/weather patterns on particular highway sections; In online retail business, of interest are often the times when customers, with diverse background, re-visit the online platforms and place their orders; In

video streaming quality control, of interest are often the modeling of the recurrent buffering processes during video streaming in a highly dynamic internet environment, and so on.

1.2. Literature, Gaps and Contributions

Statistical methods for event data have been investigated in survival analysis, reliability engineering and bioinformatics (Fleming 1991, Anderson et al. 1993, Nelson 1995, Meeker and Escobar 1998). In Liu and Pan (2020), the authors provided a comprehensive discussions on the common challenges for modeling recurrence data from a large population of heterogeneous individuals with diverse feature information. These challenges include model specification, data heterogeneity due to different operating conditions, between-individual variation (e.g., the relationship between event process and feature information may vary among individuals) and feature selection. Hence, the divide-and-conquer nature of tree-based methods is appealing when hidden sub-populations exist within a heterogeneous population. The non-parametric nature of regression trees also helps to avoid parametric assumptions on the complex interactions between event processes and features; see Liu and Pan (2020) for detailed discussions. In fact, for time-to-event data (i.e., non-recurrent events), the advantages of tree-based methods have been investigated in Huo et al. (2006), Hothorn et al. (2006), Fan et al. (2006, 2009), Chipman et al. (2010), Ishwaran et al. (2008), Ishwaran and Kogalur (2010), Ishwaran et al. (2010) and Bou-Hamad (2011).

In recent years, machine learning and Deep Recurrent Neural Network (DRNN) have received considerable attention for their advantages in capturing complex non-linear event-feature interactions (Ranganath et al. 2016, Lao et al. 2017, Wang et al. 2017, Katzman et al. 2018, Grob et al. 2018, Lee et al. 2018). These methods integrate statistical survival analysis models into the framework of deep learning without specifically addressing the existence of sub-populations with the event-feature relationship varying across sub-populations. Finally, as the volume of data grows, so does the amount of noise (irrelevant features, sampling error, measurement error, etc.); see Jordan (2019). Many features could be redundant from either the statistical modeling or domain knowledge perspective (Guyon and Elisseeff 2003, Reunanen 2003, Yuan and Lin 2005, Nilsson et al. 2007, Witten and Tibshirani 2010, Paynabar et al. 2015).

To address the challenges above, this paper proposes an additive-tree-based method for modeling recurrent event data with static and dynamic feature information. By assuming that the event process of an individual constitutes a counting process, we seek an ensemble of binary regression trees to estimate the cumulative intensity function that fully characterizes the recurrent event process. The ensemble trees are obtained under the framework of XGBoost (Chen and Guestrin 2016), and the proposed method is called **Boost-R** (**Boost**ing for **R**ecurrence Data). To our best knowledge, Boost-R is the first gradient boosted additive-tree-based statistical learning approach for modeling recurrence data with feature information. The R code is made available on GitHub (`https://github.com/dnncode/Boost-R`).

Note that, a recently proposed additive-tree-based method, known as RF-R, leverages the idea of Random Forest (RF) for modeling recurrence data (Liu and Pan 2020). The Boost-R proposed in this paper can be seen as a competitor of RF-R, just like how XGBoost is often seen as an alternative to RF (Chipman et al. 2010, Chen and Guestrin 2016). The two competitors are based on exactly opposite and independent ideas: *RF involves an ensemble of de-correlated and fully-grown trees, while Boosting fits a sequence of correlated simple trees ("weaker learners") and each tree explains only a small amount of variation*

not captured by previous trees (Freund and Schapire 1997, Chipman et al. 2010, Hastie et al. 2009). From this perspective, a natural and meaningful question to be answered is whether the gradient boosted trees can be leveraged to achieve a better modeling and prediction performance for recurrent event data. As shown in Figure 4, the relationship between Boost-R and RF-R is parallel, and the development of Boost-R in the present paper makes both RF-R and Boost-R available in practice; just like how RF and XGBoost co-exist as the two most popular off-the-shelf methods.

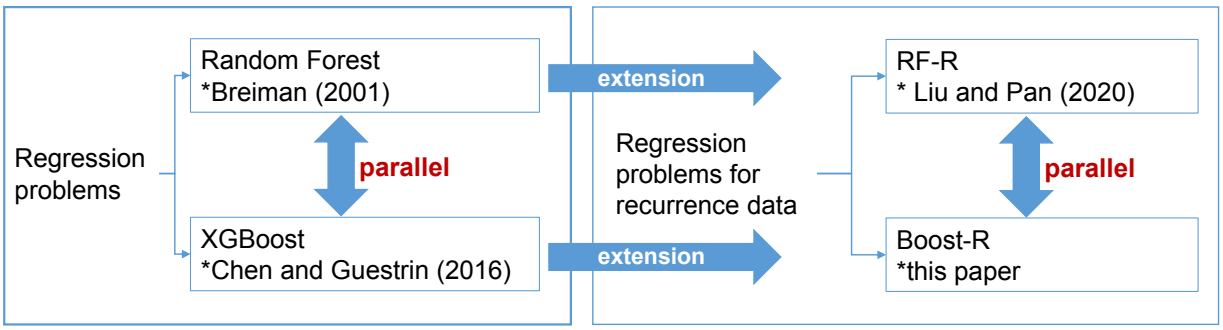

Figure 4 **The relationship between Boost-R and RF-R is similar to that between XGBoost and RF.**

The paper is organized as follows. Section 2 presents the technical details of Boost-R. Comprehensive numerical examples, including a case study, are provided in Section 3 which generate critical insights on the Boost-R algorithm and demonstrate its advantages. Section 4 concludes the paper. The paper also includes supplementary material that provides additional discussions and comparison studies using different application examples.

2. Boost-R: Boosting for Recurrence Data

2.1. The Problem Setup

Consider a population of n individuals. An individual i $(i = 1, 2, ..., n)$ experiences a sequence of r_i events at times $\boldsymbol{y}_i = (y_{i,1}, ..., y_{i,r_i}, c_i)$, where $y_{i,1}, ..., y_{i,r_i}$ are the event times and c_i is the right censoring time. Associated with individual i there exists p static features, $\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,p})$, and q time-varying dynamic features represented by a q-dimensional time series, $\boldsymbol{z}_i(t) = (z_{i,1}(t), z_{i,2}(t), ..., z_{i,q}(t))$. Hence, the recurrent event data are denoted by $\mathfrak{D} = (\boldsymbol{y}_i, \boldsymbol{x}_i, \boldsymbol{z}_i(t))$, where $|\mathfrak{D}| = n$, $\boldsymbol{x}_i \in \mathbb{R}^p$, $\boldsymbol{y}_i \in \mathbb{R}^{r_i+1}$ and $\boldsymbol{z}_i(t) : [0, \infty) \to \mathbb{R}^q$. Conditioning on $\boldsymbol{x}_i$ and $\boldsymbol{z}_i(t)$, events arising from an individual constitute a counting process, $\Lambda_i(t) : [0, \infty) \to \mathbb{N}^+$, with the cumulative intensity $\mu(t; \boldsymbol{x}_i, \boldsymbol{z}_i^h(t))$ where $\boldsymbol{z}_i^h(t) = \{\boldsymbol{z}_i(\tau); 0 \le \tau \le t\}$ is the history of the dynamic feature information associated with individual i up to t.

We seek an additive-tree-based model with K binary regression trees, $T^{(1)}, T^{(2)}, ..., T^{(K)}$, such that

$$\hat{\mu}^{(K)}(t; \boldsymbol{x}_i, \boldsymbol{z}_i^h(t)) = \sum_{k=1}^{K} T^{(k)}(\boldsymbol{x}_i, \boldsymbol{z}_i^h(t)) \tag{1}$$

where $\hat{\mu}^{(K)}(t; \boldsymbol{x}_i, \boldsymbol{z}_i^h(t))$ is the ensemble estimator of the time-dependent cumulative intensity function (for individual i) constructed from K trees. Here, the output from a tree is a time-dependent function given $\boldsymbol{x}_i$ and $\boldsymbol{z}_i^h(t)$.

Note that, there is a *major difference* between conventional binary regression trees and regression trees for recurrence data. For conventional trees, a constant is found on each tree leaf and used as the predicted value for the sub-population represented by that tree leaf. When dealing with recurrent event data, on the other hand, an individual tree performs a binary partition of the feature space, and a time-dependent function (instead of a constant) needs to be found on each tree leaf. The sum of these time-dependent functions, from multiple trees, yields the ensemble estimator of the cumulative intensity which fully characterizes the recurrent event process given feature information. This key difference requires us to devise new computationally efficient algorithms for growing the ensemble trees, $T^{(1)}, T^{(2)}, ..., T^{(K)}$, and the idea of Gradient Boosting is leveraged in this paper.

2.2. Boost-R with Static Features

We first consider a recurrent event data set with only static features, $\mathfrak{D} = (\boldsymbol{y}_i, \boldsymbol{x}_i)$, where $|\mathfrak{D}| = n$, $\boldsymbol{x}_i \in \mathbb{R}^p$ and $\boldsymbol{y}_i \in \mathbb{R}^{r_i+1}$. A conventional binary regression tree divides the feature space into D disjoint "rectangular" subspaces, R_d, $d = 1, 2, ..., D$, and each subspace is represented by a tree leaf. For any tree leaf d, a constant μ_d is found as the predicted value for individuals associated with that tree leaf. Hence, a conventional regression tree can be expressed by the linear combination of indicator functions, $T(\boldsymbol{x}; \boldsymbol{\theta}) = \sum_{d=1}^{D} \mu_d I(\boldsymbol{x} \in R_d)$, where I is an indicator function and $\boldsymbol{\theta} = (R_d, \mu_d)_{d=1}^{D}$.

For recurrent event data, on the other hand, each individual i contains a sequence of event times $\boldsymbol{y}_i$ rather than a single response value as in the classical setting of regression trees. Hence, a time-dependent function is established on each tree leaf. The sum of these functions (from multiple trees) yields the ensemble estimator of the cumulative intensity of the event process, conditioning on the features. Following this idea, we search for an additive model with K trees:

$$\hat{\mu}^{(K)}(t; \boldsymbol{x}) = \sum_{k=1}^{K} T^{(k)}(\boldsymbol{x}) = \sum_{k=1}^{K} \sum_{d=1}^{D^{(k)}} f_d^{(k)}(t) I(\boldsymbol{x} \in R_d^{(k)}). \tag{2}$$

In (2), $\hat{\mu}^{(K)}(t; \boldsymbol{x})$ is the ensemble estimator of the cumulative intensity function obtained from K trees. An individual tree is given by, $T^{(k)}(\boldsymbol{x}) = \sum_{d=1}^{D^{(k)}} f_d^{(k)}(t) I(\boldsymbol{x} \in R_d^{(k)})$, where $D^{(k)}$ is the number of leaves of tree k and $f_d^{(k)}(t)$ is a function of time. The space of trees is given by $\{T(\boldsymbol{x}; \boldsymbol{\theta}) = f_{r(\boldsymbol{x})}(t)\}$ with r denoting the mapping from $\boldsymbol{x}$ to a tree leaf.

The idea of boosting involves sequentially fitting a sequence of correlated simple trees ("weaker learners"), and each tree explains a small amount of variation not explained by previous trees. Let $\tilde{\mu}_i(t)$ be the empirical Mean Cumulative Function (MCF) estimator of the cumulative intensity function for individual i, then, a set of K trees, $T^{(k)}$, $k = 1, ..., K$, can be obtained by minimizing a regularized objective function $\mathfrak{L}^{(K)}$:

$$\text{minimize} \quad \mathfrak{L}^{(K)} = \sum_{i=1}^{n} l(\tilde{\mu}_i(t), \hat{\mu}^{(K)}(t; \boldsymbol{x}_i)) + \sum_{k=1}^{K} \Omega(T^{(k)}). \tag{3}$$

where l is a differentiable convex loss function which measures the distance between $\tilde{\mu}_i(t)$ and $\hat{\mu}^{(K)}(t; \boldsymbol{x}_i)$. Although other choices are possible, we consider the following loss function

$$l(\tilde{\mu}_i(t), \hat{\mu}^{(K)}(t; \boldsymbol{x}_i)) = \frac{1}{2} ||\tilde{\mu}_i(t) - \hat{\mu}^{(K)}(t; \boldsymbol{x}_i)||_2^2 \tag{4}$$

which is proportional to the squared L^2 distance of two time-dependent functions.

The regularization $\Omega(\cdot)$ in (3) controls the complexity of individual trees because the central idea of Boosting involves sequentially fitting a sequence of "weak learners". Hence, we consider the following regularization:

$$\Omega(T^{(k)}) = \gamma_1 D^{(k)} + \frac{1}{2}\gamma_2 \sum_{d=1}^{D^{(k)}} ||f_d^{(k)}(t)||_2^2. \tag{5}$$

The first term in (5) controls the depth (i.e., the number of leaves) of individual trees. From the perspective of Analysis of Variance (ANOVA), the value for D reflects the level of dominant interaction effects (of the features) on the recurrent event processes (Hastie et al. 2009). Hence, regularizing the depth of a tree implies that only the important main effects and lower-order interaction effects are captured. The second term in (5) adopts the shrinkage strategy in statistical learning, and penalizes the contributions from individual trees (i.e., $f_d^{(k)}(t)$) to the ensemble estimate (2).

The optimization problem (3) is a formidable combinatorial problem, but can be solved using the stagewise gradient boosting. We first discretize $\tilde{\mu}_i(t)$, $\hat{\mu}_i^{(K)}(t)$ and $f_d^{(k)}(t)$ at equally spaced times $\{t_j\}_{j=1}^m$, and let

- $\{\tilde{\mu}_{i,j}\}_{j=1}^m$ be the values of $\tilde{\mu}_i(t)$ at $\{t_j\}_{j=1}^m$;
- $\{\hat{\mu}_{i,j}^{(k-1)}\}_{j=1}^m$ be the ensemble estimates from the first $(k-1)$ trees at $\{t_j\}_{j=1}^m$;
- $\{f_{d,j}^{(k)}\}_{j=1}^m$ be the values of $f_d^{(k)}(t)$ at $\{t_j\}_{j=1}^m$.

Then, given the first $(k-1)$ trees in the ensemble, the kth tree is found by minimizing the following objective function (i.e., stagewise gradient boosting):

$$\text{minimize} \quad \sum_{i=1}^n \sum_{j=1}^m l(\tilde{\mu}_{i,j}, \hat{\mu}_{i,j}^{(k-1)} + f_{r(\boldsymbol{x}_i),j}^{(k)}\Delta) + \gamma_1 D^{(k)} + \frac{1}{2}\gamma_2 \sum_{d=1}^{D^{(k)}} ||f_{d,\cdot}^{(k)}||_2^2 \tag{6}$$

where $f_{d,\cdot}^{(k)}$ is a vector $\{f_{d,j}^{(k)}\}_{j=1}^m$, and Δ is the spacing of equally-spaced times $\{t_j\}_{j=1}^m$.

Approximating the function l in (6) by a smoother function is essential in obtaining computationally efficient gradient boosting algorithms (Hastie et al. 2009). Hence, the second-order approximation of l at $\hat{\mu}_{i,j}^{(k-1)}$ leads to

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^n \left\{ \sum_{j=1}^m l(\tilde{\mu}_{i,j}, \hat{\mu}_{i,j}^{(k-1)}) + g_{i,j} f_{r(\boldsymbol{x}_i),j}^{(k)} + \frac{1}{2} h_{i,j} (f_{r(\boldsymbol{x}_i),j}^{(k)})^2 \right\} \Delta \\ & + \gamma_1 D^{(k)} + \frac{1}{2}\gamma_2 \sum_{d=1}^{D^{(k)}} ||f_{d,\cdot}^{(k)}||_2^2 \end{aligned} \tag{7}$$

where $g_{i,j} = \partial_{\hat{\mu}_{i,j}^{(k-1)}} l(\tilde{\mu}_{i,j}, \hat{\mu}_{i,j}^{(k-1)})$ and $h_{i,j} = \partial^2_{\hat{\mu}_{i,j}^{(k-1)}} l(\tilde{\mu}_{i,j}, \hat{\mu}_{i,j}^{(k-1)})$.

Let $\Delta \to 0$ and drop the constant term $l(\tilde{\mu}_{i,j}, \hat{\mu}_{i,j}^{(k-1)})$ in (7), we have

$$\text{minimize} \quad \sum_{i=1}^n \int \left(g_i(t) f_{r(\boldsymbol{x}_i)}^{(k)}(t) + \frac{1}{2} h_i(t) (f_{r(\boldsymbol{x}_i)}^{(k)}(t))^2 \right) dt + \gamma_1 D^{(k)} + \frac{1}{2}\gamma_2 \sum_{d=1}^{D^{(k)}} ||f_{d,\cdot}^{(k)}(t)||_2^2 \tag{8}$$

where $g_i(t) = \partial_{\hat{\mu}_i^{(k-1)}(t)} l(\tilde{\mu}_i(t), \hat{\mu}_i^{(k-1)}(t))$ and $h_i(t) = \partial^2_{\hat{\mu}_i^{(k-1)}(t)} l(\tilde{\mu}_i(t), \hat{\mu}_i^{(k-1)}(t))$. Hence, once the first $(k-1)$ trees have been determined, the kth tree can be found by solving (8).

Growing the kth tree requires iteratively finding the optimal split feature and splitting points for each tree node. Naturally, the node splitting process can be terminated when the objective function (8) cannot be further reduced by splitting any of the tree nodes. For any given tree topology, let $I_d = \{i | r(\boldsymbol{x_i}) = d\}$ be a set that contains all individuals within leaf d. Then, the contribution to the objective (8) from tree node d is:

$$\int \left(g_\cdot(t) f_d(t) + \frac{1}{2}(h_\cdot(t) + \gamma_2) f_d^2(t) \right) dt + \gamma_1 \tag{9}$$

where $g_\cdot(t) = \sum_{i \in I_d} g_i(t)$ and $h_\cdot(t) = \sum_{i \in I_d} h_i(t)$. Here, the superscript $\cdot^{(k)}$ is dropped without causing confusion.

Given any candidate split feature and splitting point, tree node d can be split into two daughter nodes. Let $I_d^{(L)}$ and $I_d^{(R)}$ respectively contain the individuals in the left and right daughter nodes of d, the amount of reduction of (8) achieved by this splitting is given by:

$$\begin{aligned} G_1 = & \int \left(g_\cdot(t) f_d(t) + \frac{1}{2}(h_\cdot(t) + \gamma_2) f_d^2(t) \right) dt \\ & - \int \left(g_\cdot^{(L)}(t) f_d^{(L)}(t) + \frac{1}{2}(h_\cdot^{(L)}(t) + \gamma_2)(f_d^{(L)}(t))^2 \right) dt \\ & - \int \left(g_\cdot^{(R)}(t) f_d^{(R)}(t) + \frac{1}{2}(h_\cdot^{(L)}(t) + \gamma_2)(f_d^{(R)}(t))^2 \right) dt - \gamma_1 \end{aligned} \tag{10}$$

where $g_\cdot^{(L)}(t) = \sum_{i \in I_d^{(L)}} g_i(t)$, $h_\cdot^{(L)}(t) = \sum_{i \in I_d^{(L)}} h_i(t)$, $g_\cdot^{(R)}(t) = \sum_{i \in I_d^{(R)}} g_i(t)$ and $h_\cdot^{(R)}(t) = \sum_{i \in I_d^{(R)}} h_i(t)$. Hence, if $\max G_1 > 0$, the optimal split feature and splitting point are the ones that maximize G_1. If $\max G_1 \leq 0$, no gain can be achieved by further splitting the tree node d, making this node a terminal node. The tree growing process is terminated when no further node splitting is possible.

For a high-dimensional feature space with a large p, there is always a need arising from practice to perform feature selection. Excluding irrelevant features greatly helps to develop more accurate predictive models with improved model interpretability. By design, tree-based methods have a natural advantage in terms of feature selection. Recall that, at each tree node splitting, the optimal split variable is chosen to achieve the maximum gain in (10). Hence, the importance of a feature can be measured as the total gain achieved by splitting tree nodes based on this feature.

To make this idea formal, for any tree k in the ensemble, let $G_{d',i}^{(k)}$ be the gain achieved by splitting an internal node d' by feature i. Note that, $d' = 1, 2, \cdots, D^{(k)} - 1$ where $D^{(k)} - 1$ is the number of internal nodes of a binary tree with $D^{(k)}$ leaves. Let $s_{d',i}^{(k)} = 1$ if the internal node d' is split by feature i; otherwise $s_{d',i}^{(k)} = 0$. Then, the importance of feature i can be computed as (Hastie et al. 2009):

$$w_i = \frac{1}{K^2} \sum_{k=1}^{K} \sum_{d'}^{D^{(k)}-1} G_{d',i}^{(k)} s_{d',i}^{(k)}, \qquad \text{for } i = 1, 2, \cdots, p. \tag{11}$$

If we let $G^{(k)}_{d',i} = 1$, the importance of feature i is measured by the number of times this feature is used for splitting a node, and such a strategy appeared in Chipman et al. (2010).

Finally, the Boost-R algorithm is summarized in Algorithm 1.

Data: $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ for $i = 1, 2, ..., n$

For all $i = 1, ..., n$:
initialize $\mu^{(0)}(t) = 0$,
choose γ_1 and γ_2,
calculate $g_i(t) = \partial_{\hat{\mu}_i^{(0)}(t)} l(\tilde{\mu}_i(t), \hat{\mu}_i^{(0)}(t))$ and $h_i(t) = \partial^2_{\hat{\mu}_i^{(0)}(t)} l(\tilde{\mu}_i(t), \hat{\mu}_i^{(0)}(t))$.

for $k = 1, ..., K$ **do**
  for $d = 1, ..., D^{(k)}$ **do**
    if *d is not a terminal node* **then**
      construct I_d
      calculate $g_\cdot(t) = \sum_{i \in I_d} g_i(t)$ and $h_\cdot(t) = \sum_{i \in I_d} h_i(t)$.
      for $i = 1, ..., p$ **do**
        gain $\leftarrow 0$
        for $j = 1, ..., m$ **do**
          split the node by $x_{i,j}$
          calculate G_1 from (10)
          gain $\leftarrow \max(\text{gain}, G_1)$
        end
        if gain ≤ 0 **then**
          node d is a terminal node
        end
      end
    end
    update the tree topology
    update $\hat{\mu}^{(k)}(t) = \hat{\mu}^{(k-1)}(t) + T^{(k)}(\boldsymbol{x})$
    update $g_i(t) = \partial_{\hat{\mu}_i^{(k)}(t)} l(\tilde{\mu}_i(t), \hat{\mu}_i^{(k)}(t))$ and $h_i(t) = \partial^2_{\hat{\mu}_i^{(k)}(t)} l(\tilde{\mu}_i(t), \hat{\mu}_i^{(k)}(t))$.
  end
end

Algorithm 1: the Boost-R algorithm

2.3. Boost-R with Dynamic Features

Next, we extend Boost-R to handle both static and dynamic features. As discussed in Section 2.1, with both static and dynamic features, the recurrent event data can be denoted by $\mathfrak{D} = (\boldsymbol{y}_i, \boldsymbol{x}_i, \boldsymbol{z}_i(t))$, where $|\mathfrak{D}| = n$, $\boldsymbol{x}_i \in \mathbb{R}^p$, $\boldsymbol{y}_i \in \mathbb{R}^{r_i+1}$ and $\boldsymbol{z}_i(t) : [0, \infty) \to \mathbb{R}^q$.

Incorporating dynamic features into a tree-based method is challenging especially when dynamic features have cumulative effects on the event process (Bacchetti and Segal 1995, Bou-Hamad et al. 2009). In other words, for individual i, its cumulative intensity depends on the entire history of the dynamic feature information, $\boldsymbol{z}_i^h(t) = \{\boldsymbol{z}_i(\tau); 0 \leq \tau \leq t\}$, associated with that individual. One approach, proposed in Liu and Pan (2020), is to split a tree node based on static features, while the data on each node are modeled by a separate model that explains the effects of dynamic features. In other words, dynamic features

are nested with static features. At each tree leaf, different individuals are associated with different time-dependent functions. This approach can be justified when sub-populations are mainly characterized by static attributes, while dynamic features are used for explaining the between-individual variation within a tree node (i.e., the variation between event processes for individuals sharing similar attributes).

Following the idea above, an additive-tree-based model with both static and dynamic features can be expressed as:

$$\hat{\mu}^{(K)}(t;\boldsymbol{x}_i,\boldsymbol{z}_i^h(t))=\sum_{k=1}^{K}T^{(k)}(\boldsymbol{x}_i,\boldsymbol{z}_i^h(t))=\sum_{k=1}^{K}\sum_{d=1}^{D^{(k)}}f_d^{(k)}(t;\boldsymbol{z}_i^h(t))I(\boldsymbol{x}_i\in R_d^{(k)}) \tag{12}$$

where $\hat{\mu}^{(K)}(t;\boldsymbol{x}_i,\boldsymbol{z}_i^h(t))$ is the ensemble estimator of the cumulative intensity for individual i based on K trees; and $T^{(k)}(\boldsymbol{x}_i,\boldsymbol{z}_i^h(t))=\sum_{d=1}^{D^{(k)}}f_d^{(k)}(t;\boldsymbol{z}_i^h(t))I(\boldsymbol{x}_i\in R_d^{(k)})$ with $D^{(k)}$ denoting the number of leaves of tree k and $f_d^{(k)}(t;\boldsymbol{z}_i^h(t))$ being a time-dependent function depending on $\boldsymbol{z}_i^h(t)$. It is easy to see how the idea of Liu and Pan (2020) is embedded into (12): a tree node is split based on static features, and the data on each node are modeled by a second model that incorporates the dynamic features.

The same idea behind Algorithm 1 can be adopted and modified to construct the boosting trees (12) with dynamic features. However, before we present the extended algorithm, some necessary modifications are needed:

- Because it is less realistic to assume a parametric form for $f_d^{(k)}(t;\boldsymbol{z}_i^h(t))$ in (12), we adopt the non-parametric approach and model $f_d^{(k)}(t;\boldsymbol{z}_i^h(t))$ as

$$\begin{aligned}f_d^{(k)}(t;\boldsymbol{z}_i^h(t))&=\sum_{l}^{q}\int_0^t b_l(z_{i,l}(\tau);\boldsymbol{\beta}_l^{(d)})d\tau\\&=\sum_{l}^{q}\sum_{j}^{u+v}\left(\beta_{j,l}^{(d)}\int_0^t B_{j,v}(z_{i,l}(\tau))d\tau\right)\end{aligned} \tag{13}$$

where b_l is a linear combination of B-splines bases, $\boldsymbol{\beta}_l$ are the control or de Boor points, $B_{j,v}$ is the jth B-splines basis function of order v, and u is the number of internal knots.

- The regularization term (5) needs to be modified. Note that, for Boost-R with only static features, all individuals on a node d share the same function $f_d(t)$. However, when dynamic features are included, individuals on the same node (i.e., individuals share the same static feature $\boldsymbol{x}$) are associated with different $f_d(t;\boldsymbol{z}^h(t))$ because these individuals typically possess different history of dynamic feature information. Hence, it is no longer meaningful to use the shrinkage strategy by penalizing $||f_d(t;\boldsymbol{z}^h(t))||_2^2$, which depends on dynamic features. This consideration motivates us to adopt the idea of Group Lasso, and obtain a set of K trees by minimizing

$$\text{minimize}\quad \mathfrak{L}^{(K)}=\sum_{i=1}^{n}l(\tilde{\mu}_i(t),\hat{\mu}^{(K)}(t;\boldsymbol{x}_i,\boldsymbol{z}_i^h(t)))+\gamma_1\sum_{k=1}^{K}D^{(k)}+\frac{1}{2}\gamma_2\sum_{d=1}^{D^{(k)}}\sum_{l}^{q}||\boldsymbol{\beta}_{\cdot,l}^{(d)}||_2. \tag{14}$$

where $\boldsymbol{\beta}_{\cdot,l}^{(d)}=(\beta_{1,l}^{(d)},\beta_{2,l}^{(d)},\cdots,\beta_{u+v,l}^{(d)})$ for $l=1,2,\cdots,q$. The last term in (14) borrows the idea from Group Lasso, which has been widely used for model selection with grouped variables (Yuan and Lin 2007). In (14), $||\boldsymbol{\beta}_{\cdot,l}^{(d)}||_2$ measures the size of $\boldsymbol{\beta}_{\cdot,l}$ which corresponds to dynamic feature l, i.e., the lth group in Group Lasso. If the lth feature turns out to be less important, it is necessary to drop the entire group vector, $\boldsymbol{\beta}_{\cdot,l}$, on a tree node.

• The node splitting procedure needs to be modified. In Section 2, (9) and (10) are obtained from (8) because all individuals on node d (i.e. for all $i \in I_d$) share the same function $f_d(t)$. For the same reason, $g_\cdot(t) = \sum_{i \in I_d} g_i(t)$ and $h_\cdot(t) = \sum_{i \in I_d} h_i(t)$ can be properly defined. When dynamic features are included, individuals on the same node are not associated with the same $f_d(t; \boldsymbol{z}^h(t))$ as individuals are associated with different dynamic features. As a result, $g_\cdot(t)$ and $h_\cdot(t)$ cannot be defined and the "gain" in (10) needs to be modified as:

$$G_2 = F_2(I_d) - F_2(I_d^{(L)}) - F_2(I_d^{(R)}) - 2\gamma_1 \tag{15}$$

where

$$F_2(I) = \sum_{i \in I} \left\{ \int_0^{c_i} g_i(t) f_d(t; \boldsymbol{z}_i^h(t)) + \frac{1}{2} h_i(t) f_d^2(t; \boldsymbol{z}_i^h(t)) dt \right\}. \tag{16}$$

Based on the discussions above, the extended Boost-R algorithm with both static and dynamic features is summarized in Algorithm 2.

Data: $(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}_i(t))$ for $i = 1, 2, ..., n$

For all $i = 1, ..., n$:
initialize $\mu^{(0)}(t) = 0$,
choose γ_1 and γ_2,
calculate $g_i(t) = \partial_{\hat{\mu}_i^{(0)}(t)} l(\tilde{\mu}_i(t), \hat{\mu}_i^{(0)}(t))$ and $h_i(t) = \partial^2_{\hat{\mu}_i^{(0)}(t)} l(\tilde{\mu}_i(t), \hat{\mu}_i^{(0)}(t))$.

for $k = 1, ..., K$ **do**
  for $d = 1, ..., D^{(k)}$ **do**
    if *d is not a terminal node* **then**
      construct I_d
      for $i = 1, ..., p$ **do**
        gain $\leftarrow 0$
        for $j = 1, ..., m$ **do**
          split the node by $x_{i,j}$
          calculate G_2 from (15)
          gain $\leftarrow \max(\text{gain}, G_2)$
        end
        if gain ≤ 0 **then**
          node d is a terminal node
        end
      end
    end
    update the tree topology
    update $\hat{\mu}^{(k)}(t) = \hat{\mu}^{(k-1)}(t) + T^{(k)}(\boldsymbol{x})$
    update $g_i(t) = \partial_{\hat{\mu}_i^{(k)}(t)} l(\tilde{\mu}_i(t), \hat{\mu}_i^{(k)}(t))$ and $h_i(t) = \partial^2_{\hat{\mu}_i^{(k)}(t)} l(\tilde{\mu}_i(t), \hat{\mu}_i^{(k)}(t))$.
  end
end

Algorithm 2: Boost-R algorithm with both static and dynamic features

3. Numerical Examples, R Code and Applications

Numerical studies, including a case study, are presented in this section to generate some critical insights on how Boost-R performs and illustrate the applications of Boost-R.

3.1. Computer Code

Boost-R has been implemented in R and leverages the parallel computing capabilities of R. The code is available at GitHub (`https://github.com/dnncode/Boost-R`), and the use of the R code is demonstrated throughout this section.

3.2. Investigate the basic properties using `DATASET A` and `DATASET B`

To develop some basic understanding of how Boost-R performs, we start with a simple numerical example involving only 200 individuals. For each individual, two static features are respectively sampled from the unit interval $[0, 1]$. Let $x_{i,j}$ denote the value of feature j associated with individual i ($i = 1, 2, ..., 200$, $j = 1, 2$), the recurrent events of individual i are simulated from a homogeneous Poisson process with the following intensity:

$$\lambda_i = \begin{cases} 0.01 & \text{if } 0 \leq x_{i,1}, x_{i,2} \leq 0.5 \\ 0.10 & \text{if } 0.5 < x_{i,1}, x_{i,2} \leq 1 \\ 0.05 & \text{otherwise} \end{cases} \tag{17}$$

This data set is referred to as `DATASET A` in this paper.

To illustrative the Boost-R algorithm, we start with some arbitrarily chosen values: $K = 50$, $\gamma_1 = 300$ and $\gamma_2 = 100$. In the R code, the boosting trees are grown using the function `BoostR`:

```
BoostR.out = BoostR(data, X, K.value=50,
                    gamma1.value=300, gamma2.value=100, D.max=4)
```
(18)

where `data` contains the recurrent event data, `X` is a matrix that contains feature information, `K.value`, `gamma1.value` and `gamma2.value` are the specified values for K, γ_1 and γ_2, and `D.max` triggers the termination of the tree growing process once the number of leaves of a tree is not smaller than `D.max`. The function `BoostR` returns an object `BoostR.out`. Four R functions have been created to visualize the output of Boost-R. These functions include `Plot_Partition`, `Plot_Individual`, `Plot_Leaf`, and `Plot_Imp`, which will be discussed next.

The function, `Plot_Partition(BoostR.out)`, visualizes the binary partitions by individual trees, as well as $f_d^{(\cdot)}(t)$ on each partition. Figure 5 shows the first 10 trees obtained from the Boost-R algorithm. Columns 1 and 3 of this figure present the binary partitions of the feature space $[0, 1]^2$ by individual trees. Columns 2 and 4 show the contribution (i.e., $f_d^{(\cdot)}(t)$) to the estimated cumulative intensity function from each tree leaf.

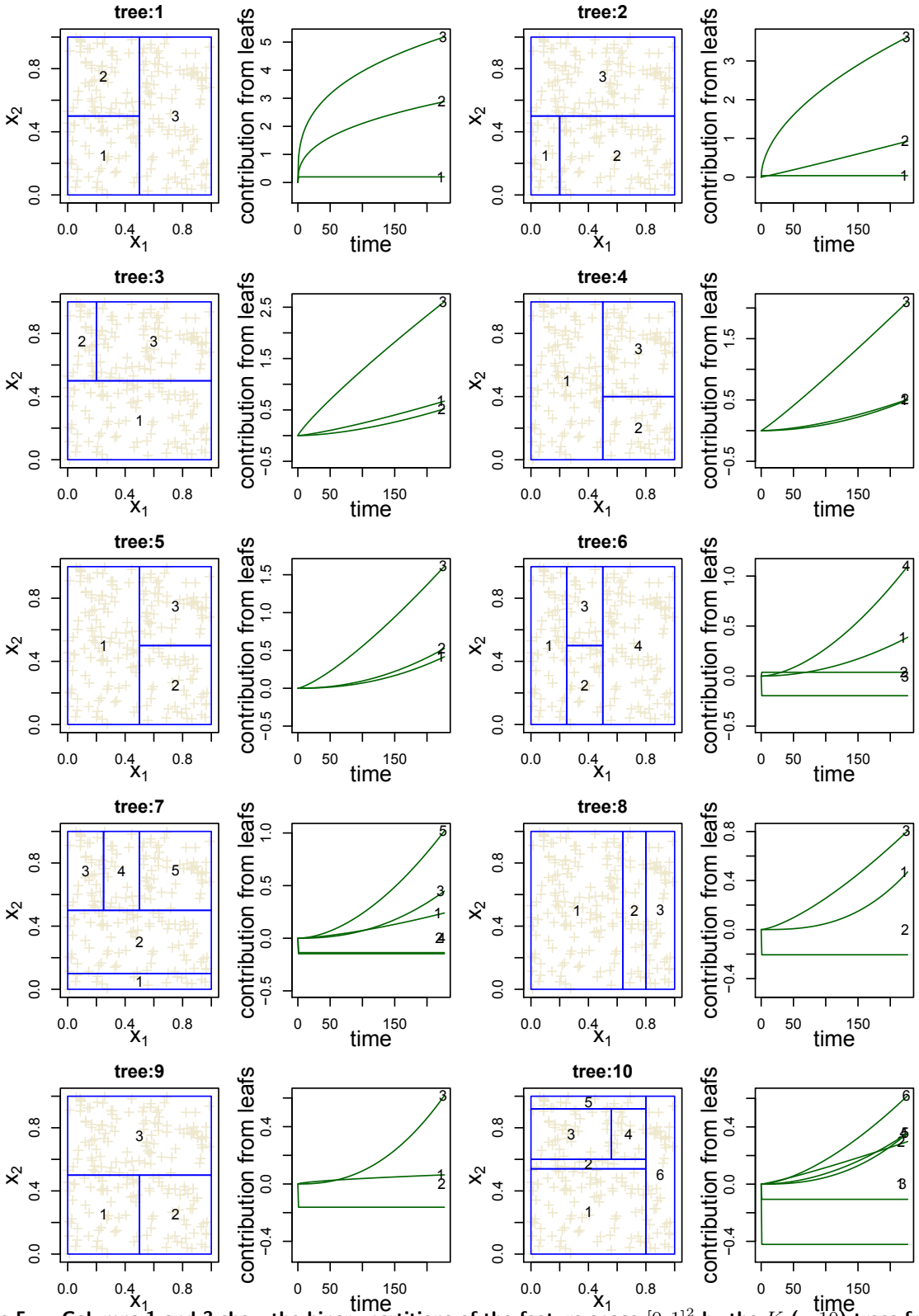

Figure 5 **Columns 1 and 3 show the binary partitions of the feature space $[0,1]^2$ by the K $(=10)$ trees from the ensemble. Columns 2 and 4 show the contribution (i.e., $f_d^{(\cdot)}(t)$) to the estimated cumulative intensity function from each tree leaf of a tree.**

It is seen from Figure 5 that each tree performs a binary partition of the feature space, which divides the 200 individuals into several sub-populations represented by tree leaves. For each sub-population, the contribution to the cumulative intensity function $f_d^{(\cdot)}(t)$ is computed. Three critical observations are obtained:

• The idea behind boosting suggests that a new tree is added to explain only a small amount of variation not captured by previous trees. Hence, the newly added tree is used to perform some necessary adjustments to the output generated from previous trees (i.e., performance boosting). By examining the scale of the vertical axis in columns 2 and 4, it is easy to see that the amount of adjustment by a newly added tree becomes smaller as more trees have already been included in the ensemble. Using `Plot_Individual(BoostR.out)`, Figure 6 shows how the final ensemble estimates for individuals 1 and 2 are obtained by aggregating outputs from individual trees.

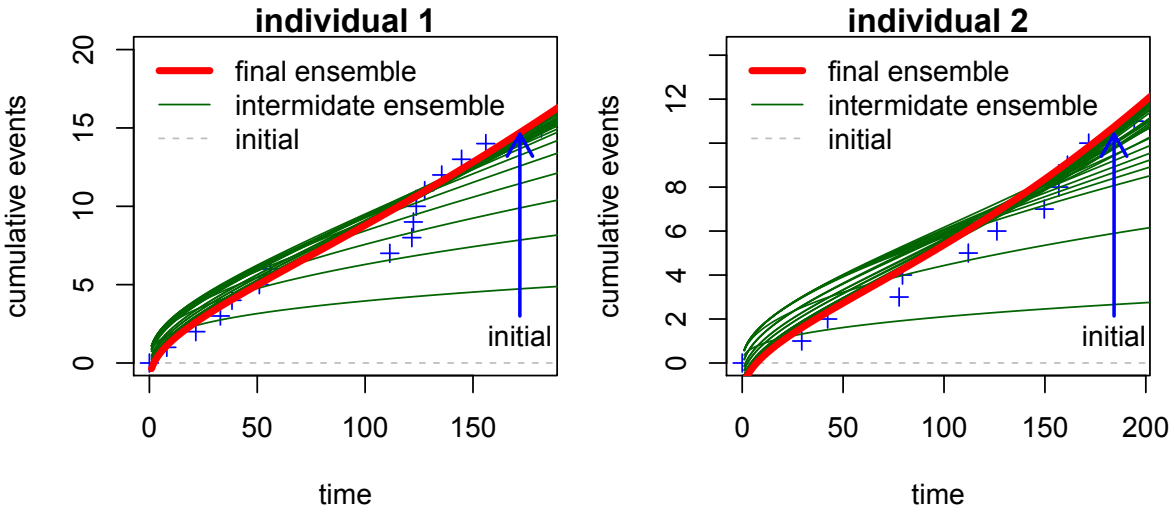

Figure 6 **The ensemble estimate of cumulative intensity functions by adding the contributions from individual boosting trees. The blue arrow shows how the initial cumulative intensity function (the horizontal dash line) converges to the ensemble estimate (the thick red curve) as more trees are grown.**

• The amount of adjustment made by a newly added tree is not necessarily positive; for example, on leaves 2 and 4 of tree #7. Typically, this happens when the estimated cumulative intensity is getting closer to the true function. Under such a circumstance, a newly added tree may suggest either increase or decrease the estimated cumulative intensity for certain sub-populations.

• The key idea behind boosting trees is that each individual tree must be kept simple to form weak learners. In Boost-R, two mechanisms are used to regularize the tree complexity, i.e., the regularization term (5) and `D.max` in (18) specified by users. Figure 7, generated by `Plot_Leaf(BoostR.out)`, shows the number of leaf nodes for the 50 trees. It is interesting to see that: (i) Since the maximum number of tree leaves, `D.max`, is set to 4 in this example, the tree growing process is forced to stop once the number of leaves has exceeded 4 (note that, the final number of leaves is not necessarily 4 in this case if two or more nodes are split simultaneously before the algorithm ends). As shown in Figure 7, this rule applies to tree #7, #10, #15 and #17; (ii) For all other trees, the tree growing processes are terminated before the number of leaves has reached `D.max`. This observation justifies the effectiveness of the regularization (5) on tree complexity; (iii) After tree #28, the remaining trees consist of only the root node. In this example, as most of the variation has been

effectively explained by the first 28 trees, the gain of adding a new tree to the ensemble is outweighed by the penalty incurred by adding that new tree. Note that, it would not be possible to observe such a phenomenon if the regularization term (5) was removed (as there would be no penalty associated with increasing tree complexity). From this perspective, Figure 7 also provides some insights on the choice of K, which will be investigated later.

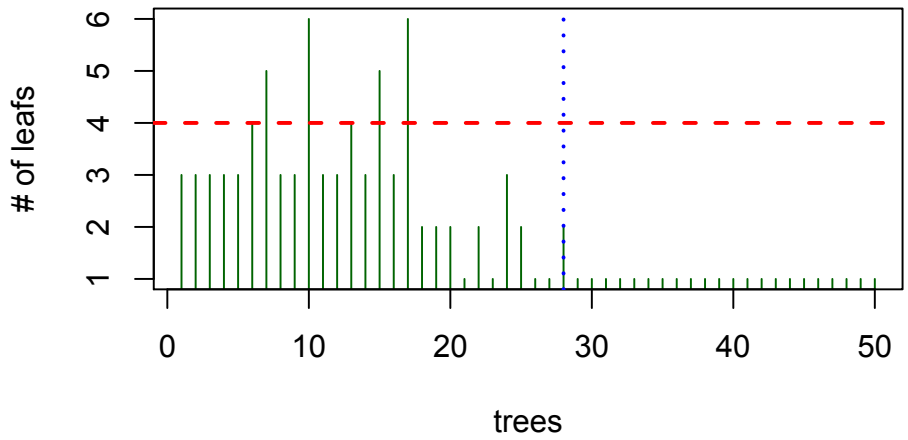

Figure 7 **Number of leaf nodes for the 50 trees. Here, we purposely overfit the data in order to generate some critical insights on how Boost-R performs.**

The boosting trees obtained from Boost-R are capable of accurately capturing the interactions between the recurrent event process and features. Figure 8 shows both the actual (left panel) and estimated (right panel) relationship between the cumulative intensity function and features. Note that, since the Boost-R algorithm does not assume that the intensity function is time-invariant, the intensity shown on the right panel is the average intensity over time over the feature space. Figure 8 clearly demonstrates the potential of Boost-R in capturing the relationship between recurrent event processes and features. More complex scenarios are investigated in Section 3.3.

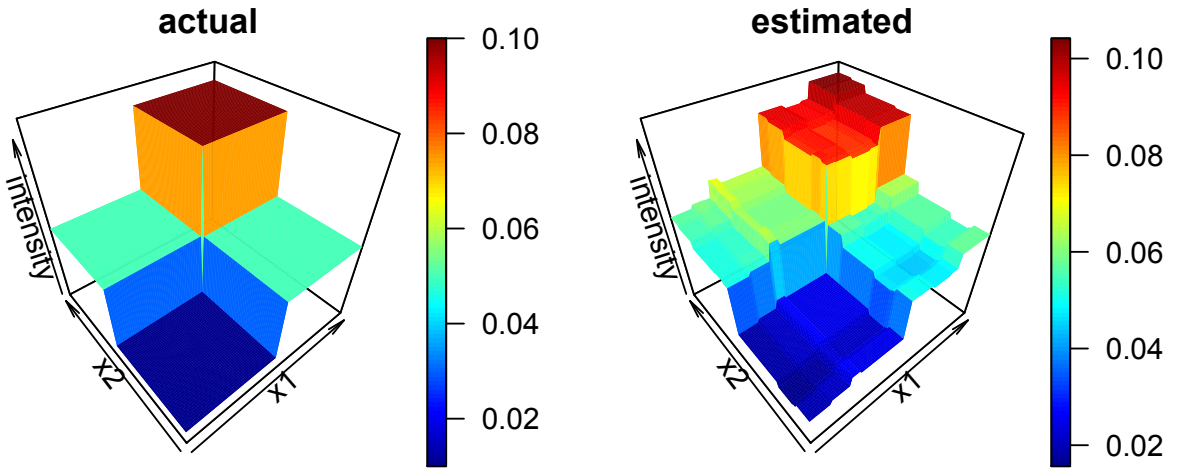

Figure 8 **Learning the relationship between the cumulative intensity function and features. Left panel: the true relationship; Right panel: the estimated relationship by Boost-R.**

Boost-R requires one to specify the parameters, γ_1, γ_2 and K. The first two parameters γ_1 and γ_2 control the complexity of individual trees, while K determines the number of trees in the ensemble. In general, K needs to be sufficiently large so that there will be enough number of trees, but not too large which causes overfitting.

As a common strategy in practice, we explore the suitable values for γ_1 and γ_2, leaving K as the primary parameter (Hastie et al. 2009). Although it is theoretically possible to perform a grid search for the best combinations of γ_1 and γ_2 on a two-dimensional space, such an approach may not be practical nor necessary in practice when it is computationally intensive to run Boost-R on big datasets. Hence, we resort to a powerful tool in computer experiments—the space-filling designs (Joseph 2016). The idea of space-filling designs is to have points everywhere in the experimental region with as few gaps as possible, which serves our purpose very well. The top left panel of Figure 9 shows the Maximum Projection Latin Hypercube Design (MaxProLHD, Joseph et al. (2015)) of 15 runs with different combinations of γ_1 and γ_2, where the experimental ranges for these two parameters are respectively $[0, 600]$ and $[0, 200]$. The top right panel of Figure 9 shows the box plot of the number of tree leaves per tree in an ensemble, for each combination of γ_1 and γ_2. Since the key idea behind boosting trees is that each individual tree needs to be kept simple with 4 to 8 leaves (Hastie et al. 2009), we quickly identify that Designs #3, #4 and #6 provide the most suitable combinations of γ_1 and γ_2. From the top left panel of Figure 9, these three design points are adjacent to each other, indicating that the appropriate choices for γ_1 and γ_2 are respectively within $[75, 220]$ and $[75, 175]$. If necessary, a more refined search can be perform in a much smaller experimental region.

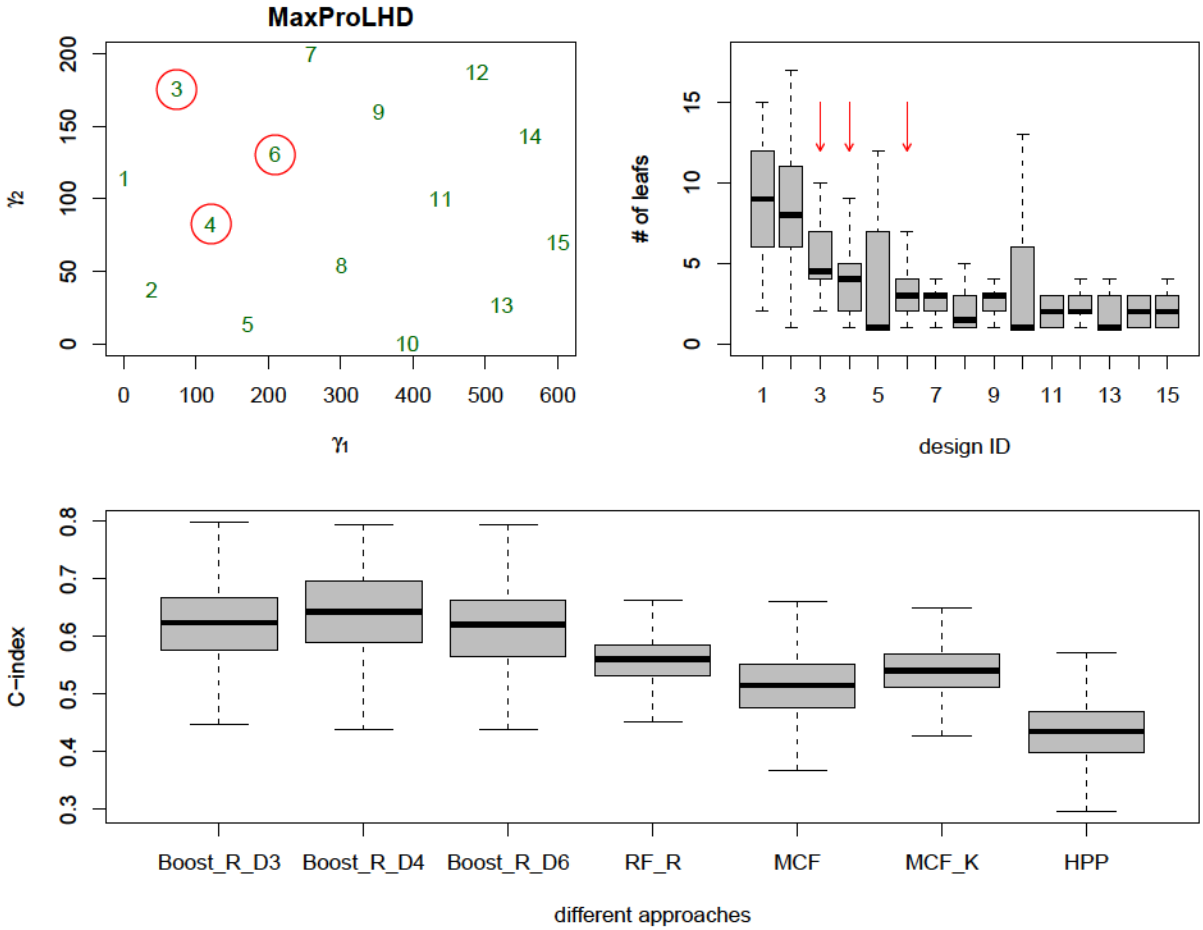

Figure 9 **Model improvements and comparison. Top left panel: the MaxProLHD design for 15 combinations of γ_1 and γ_2; Top right panel: the number of tree leaves for trees in an ensemble; Bottom: comparison of C-index for different models**

We compare the performance of Boost-R with that of existing approaches, including 1) *RF-R*: Random Forest for Recurrence Data; 2) *MCF*: the nonparametric estimation for Mean Cumulative Function (MCF) without using feature information; 3) *MCF-K*: the nonparametric estimation for MCF utilizing only the data from the K nearest neighbors of an individual (the distance between two individuals is defined by the Euclidean distance in the feature space); and 4) *HPP*: the HPP model with a log-linear intensity of static features.

Cross-validation is used to evaluate the performance of all candidate approaches. For each approach, the data set is randomly divided into a training set (150 individuals) and a testing set (50 individuals). The model is trained using the training set, and the prediction C-index is used as the performance measure. The above procedure is repeated for 500 times in order to generate the boxplot of C-indices at the bottom of Figure 9. It is seen that, the Boost-R (based on the three best designs, Designs #3, #4 and #6), yields a higher C-index than other competing methods. Here, the C-index, or Harrell's concordance index, was firstly proposed in Harrell et al. (1982) for evaluating the amount of information a medical test provides about individual patients. For the problem considered in this paper, the C-index can be interpreted as the empirical probability of correctly ranking any two individuals in terms of their cumulative number of failures, and can be calculated in the

following way: 1) form all pairs of individuals from the testing data set. 2) for each pair, rank the two individuals based on the cumulative number of events up to a given time. 3) for each pair, rank the two individuals based on the predicted cumulative number of events up to the same time. If the predicted rankings are consistent with the observed rankings, let $C_i = 1$ for pair i, otherwise $C_i = 0$; and 4) the C-index for a testing data set is the empirical probability of correctly ranking any two individuals.

Finally, to illustrate the feature selection capability of Boost-R, we include eight randomly generated redundant covariates, $\{x\}_{i=3}^{10}$, in `DATASET A`. The new data set with redundant covariates is referred to as `DATASET B`. We re-run Boost-R using `DATASET B`, and the function, `Plot_Imp(BoostR.out, standardize=TRUE)`, shows the importance for the 10 features as defined in (11); see Figure 10. Here, we standardize the importance measure (11) such that the highest and lowest importance are respectively 1 and 0. It is immediately seen that the algorithm successfully identify the correct features, x_1 and x_2.

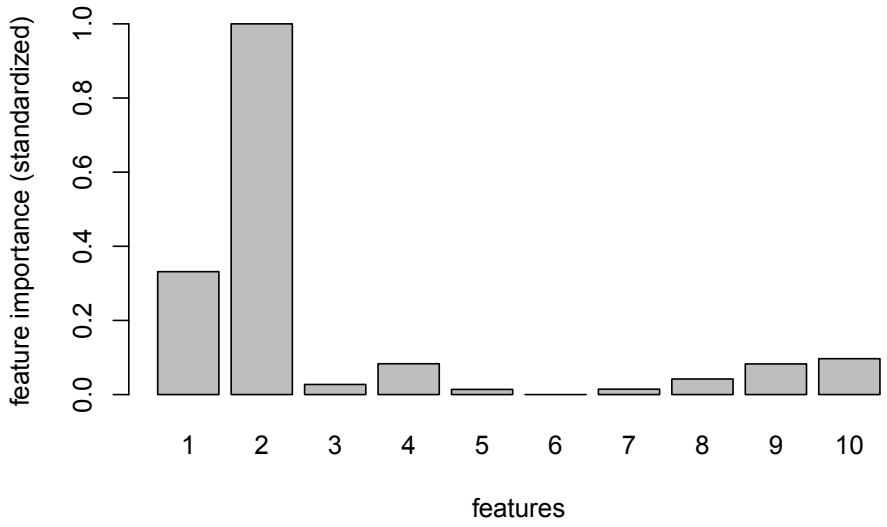

Figure 10 **Standardized feature importance measured by the total gain achieved by splitting tree nodes based on a given feature; see (11)**

3.3. More complicated scenarios using `DATASET C` and `DATASET D`

In the previous illustrative example, `DATASETS A` and `B` are simulated from a simple HPP with intensity (17). To investigate the learning capabilities of Boost-R, we consider more complicated interactions between event processes and features. In particular, `DATASET C` and `DATASET D` respectively consist of the simulated recurrent event data from 1000 individuals. For both datasets, two features, x_1 and x_2, are sampled from the unit interval $[0, 1]$. In addition,

- For `DATASET C`, the event times for individual i are simulated from a non-homogeneous Poisson process using the thinning method (Lewis and Shedler 1979) with the following intensity function:

$$\lambda_i(t) = \begin{cases} 1.5t^{-0.5} & \text{if } (x_{i,1}-0.5)^2 + (x_{i,2}-0.5)^2 \le 0.04 \\ t^{-0.5} & \text{if } 0.04 < (x_{i,1}-0.5)^2 + (x_{i,2}-0.5)^2 \le 0.16 \\ 0.5t^{-0.5} & \text{otherwise} \end{cases} \tag{19}$$

• For `DATASET D`, the event times for individual i are simulated from a non-homogeneous Poisson process with the following intensity function:

$$\lambda_i(t) = 0.01t^{0.5} \exp(0.5(x_{i,1} - 0.5)^2 + 2(x_{i,2} - 0.5)^2). \tag{20}$$

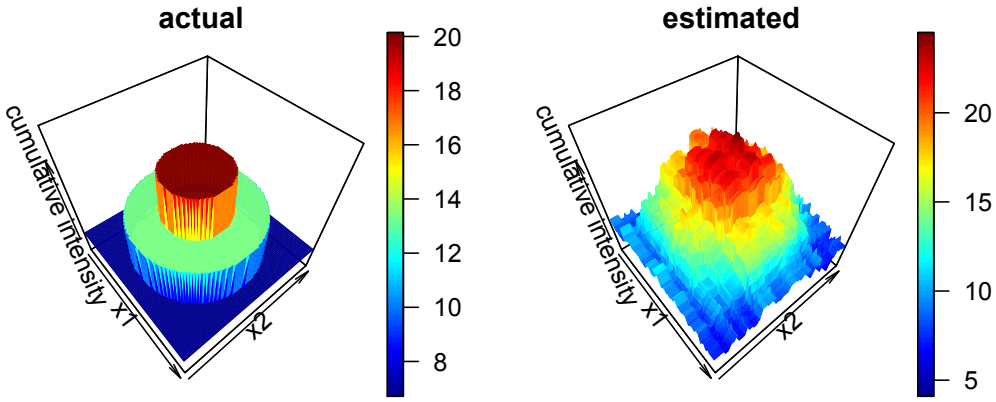

Figure 11 **Actual (left) and estimated (right) cumulative intensity at time 50 based on** `DATASET C`

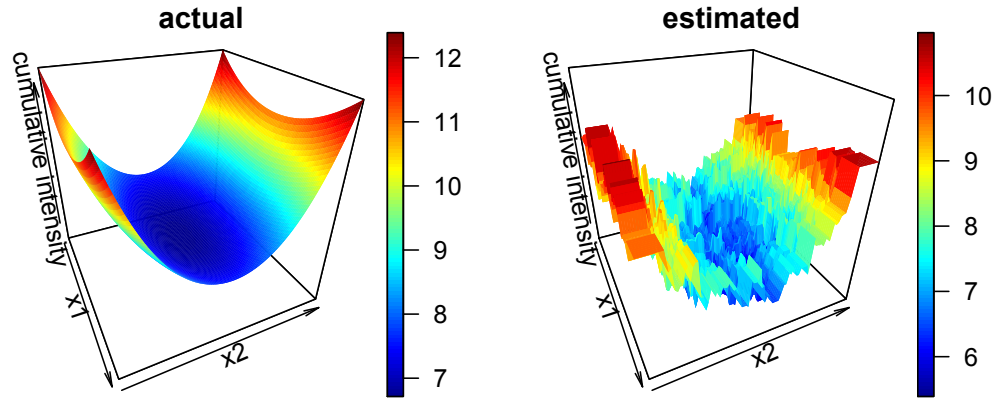

Figure 12 **Actual (left) and estimated (right) cumulative intensity at time 100 based on** `DATASET D`

We run the Boost-R algorithms using both data sets. For `DATASET C`, we let $K = 500$, $\gamma_1 = 10$ and $\gamma_2 = 5$. For `DATASET D`, we let $K = 300$, $\gamma_1 = 100$ and $\gamma_2 = 100$. Based on `DATASET C`, Figure 11 shows both the actual cumulative intensity (left panel) and estimated cumulative intensity (right panel) at time 50 over the feature space. Based on `DATASET D`, Figure 12 shows both the actual cumulative intensity (left panel) and estimated cumulative intensity (right panel) at time 100 over the feature space. It is seen that, Boost-R successfully captures the complex relationship between cumulative intensity and features, as specified in (19) and (20). Such complicated and highly nonlinear relationships can hardly be specified (unless they are known in advance) when traditional parametric approaches are used.

3.4. Modeling the Failure Processes of Oil and Gas Wells

To run Boost-R with both static and dynamic features (i.e., Algorithm 2), we model $f_d^{(k)}(t; \boldsymbol{z}_i(t))$ in (13) by cubic splines with two internal knots, and let $K = 300$, $\gamma_1 = 300$, $\gamma_2 = 100$, $v = 3$ and $u = 2$. In our R code, the boosting trees are grown using the function `BoostR2`:

```
BoostR.out = BoostR2(data, X, Z=z.list, K.value=300,
          gamma1.value=300, gamma2.value=100, u.value=2, v.value=3, D.max=4)
```
(21)

where `data` contains the recurrent event times, `X` is a matrix that contains the static feature information, Z contains the dynamic feature information, `K.value`, `gamma1.value`, `gamma2.value`, `u.value` and `v.value` are respectively the specified values for K, γ_1, γ_2, u and v, and the last input `D.max` determines the termination of the tree growing process once the number of leaves per tree reaches or exceeds `D.max`. The function `BoostR2` returns an objective that contains the output of Boost-R with both static and dynamic features, i.e., Algorithm 2. The output generated by `BoostR2` can be visualized by the following functions: `Plot_Partition`, `Plot_Individual`, `Plot_Leaf`, `Plot_Imp`, and `Plot_Interaction`.

Figure 13 is generated by `Plot_Imp(BoostR.out, standardize=TRUE)`, and shows the importance of the eight static system attributes as the total gain respectively achieved by splitting tree nodes based on each feature. The algorithm clearly identifies x_7 and x_8, the geo-locations, as the two most important system attributes. Because wells at similar geographical locations share common, but unknown, environmental conditions (e.g., temperature and humidity variation, soil type, contamination, etc.), geo-locations serve as important proxies in capturing those unknown environmental factors which may lead to some important spatial patterns such as trend and clustering. The results shown in Figure 13 confirm that these unknown environmental factors significantly influence the system failure processes, leading to different failure patterns among these well systems.

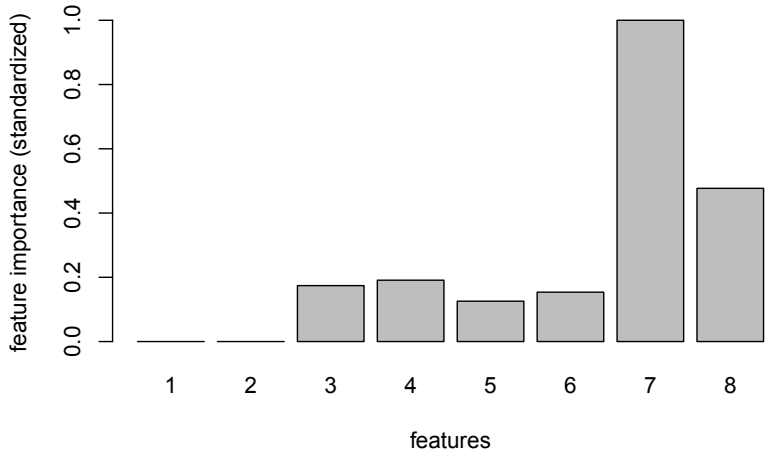

Figure 13 **Feature importance for the 8 static well attributes, which is measured by the total gain achieved by splitting tree nodes based on a given feature; see (11)**

Next, we re-run the Boost-R algorithm by retaining the two static features, x_7 and x_8, and the dynamic gearbox torque. Although one might as well include $x_3, x_4, ..., x_6$, keeping only x_7 and x_8 allows us to effectively visualize the interesting interaction between the estimated B-splines coefficients $\boldsymbol{\beta}_{\cdot,1}$ and spatial locations x_7 and x_8.

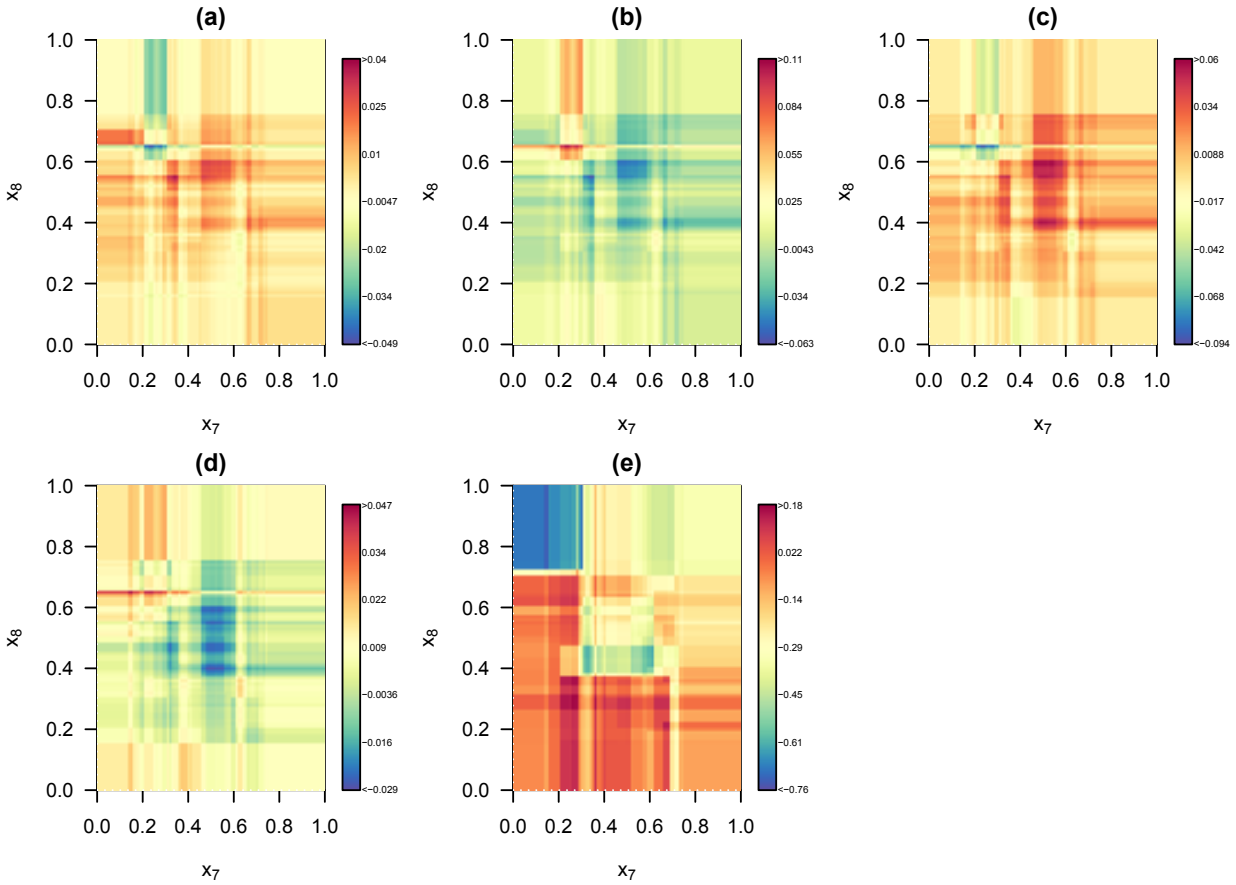

Figure 14 **Aggregated B-splines coefficients over the spatial domain. Subplots (a) to (e) respectively correspond to the coefficients $\beta_{1,1}$, $\beta_{2,1}$, $\cdots$, $\beta_{5,1}$.**

Because cubic splines with two internal knots are used in this example, we have $\boldsymbol{\beta}_{\cdot,1} = (\beta_{1,1}, \beta_{2,1}, \cdots, \beta_{5,1})$. Figure 14, which is generated by `Plot_Interaction(BoostR.out)`, provides a spatially aggregated view of the five estimated B-splines coefficients over the spatial domain. Note that, each boosting tree partitions the spatial domain $[0,1]^2$ into several rectangular areas and the estimated value of $\boldsymbol{\beta}_{\cdot,1}$ is obtained for each area. Because $\boldsymbol{\beta}_{\cdot,1}$ can be viewed as the effects of the dynamic feature on the recurrent event processes, it is immediately seen that such effects vary over the spatial domain. In other words, the cumulative failure intensities at different geo-locations are influenced by the operational conditions. For example, the aggregated values of $\beta_{1,1}$ appears to be lower in the area where $0.2 < x_7 < 0.25$ and $x_8 > 0.6$, while the aggregated value of $\beta_{2,1}$ is larger in approximately the same area. This observation strongly demonstrates the effectiveness of Boost-R in capturing the interactions between static and dynamic features, by leveraging the advantages of binary tree structures.

Using `Plot_Individual(BoostR.out)`, Figure 15 shows the estimated cumulative failure intensity and cumulative failure counts of four selected well systems (left column). The observed gearbox torque for these well systems are also shown in the right column. We see that, Boost-R successfully estimates the cumulative failure intensity for heterogeneous individuals with diverse system attributes (static features) and operating conditions (dynamic feature). Such an observation is encouraging and demonstrates the potential of Boost-R for recurrent event data analytics: the system heterogeneity is addressed by the "divide-and-conquer" structure of binary trees, and the non-parametric approaches (including the

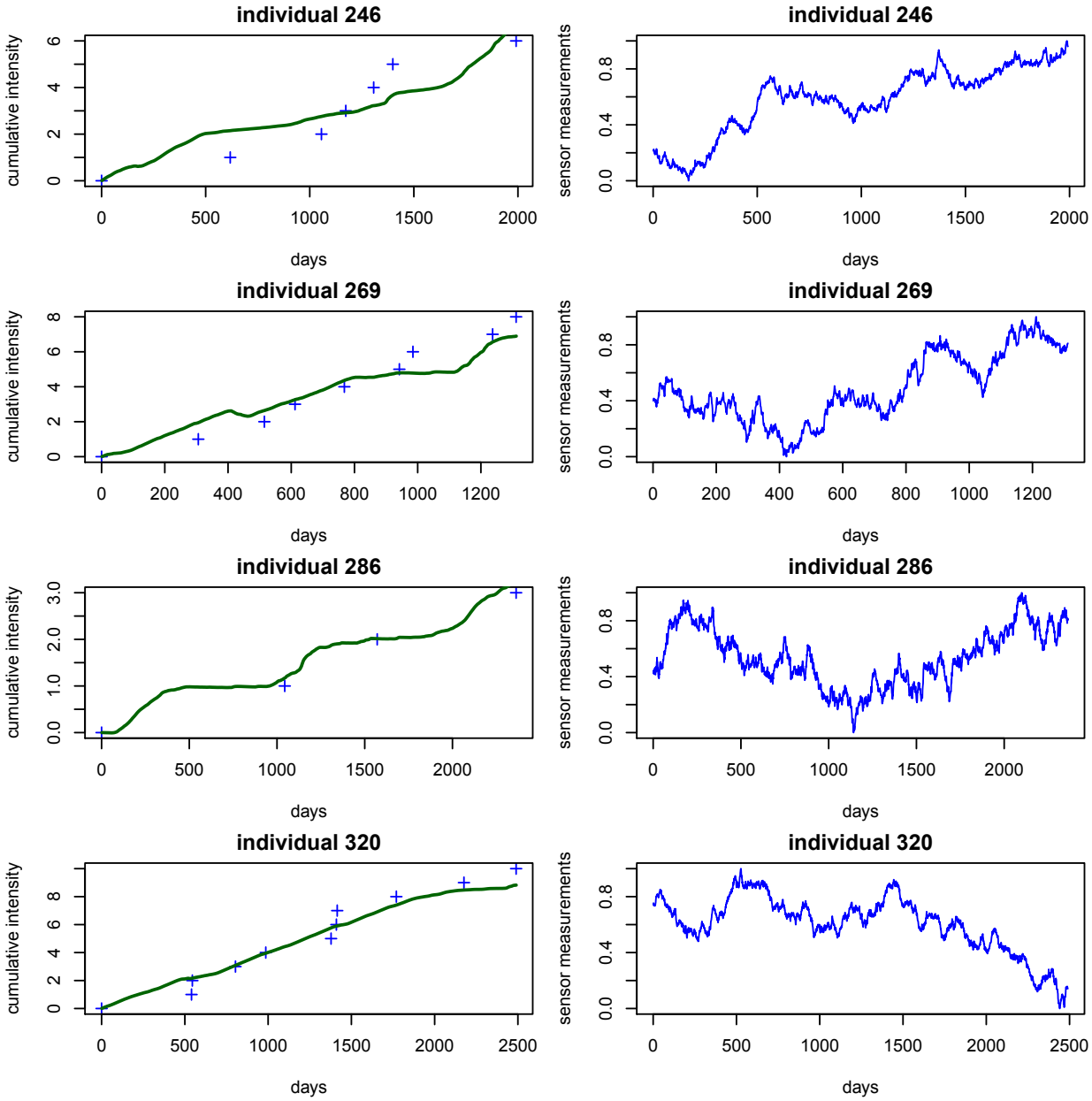

Figure 15 **Estimated cumulative intensity (left) and gearbox torque (right) for selected well systems.**

binary tree and B-splines) are used to capture the complex, often non-linear, interactions between recurrent event processes and feature information without imposing parametric assumptions.

In the Appendices, we provide additional application examples and comparison studies between Boost-R and other methods. In particular, we also provide some discussions on the potential use of XGBoost for recurrent event data.

4. Conclusions

This paper proposed an additive-tree-based statistical learning approach, known as Boost-R (**Boost**ing for **R**ecurrence Data), for modeling recurrent event data with both static and

dynamic feature information. The technical details behind Boost-R have been presented. Gradient boosting algorithms have been developed to obtain an ensemble of correlated trees that generate the estimated cumulative intensity functions characterizing the recurrent event processes given feature information. To our best knowledge, Boost-R is the first gradient boosted additive-tree-based model for recurrence data with both static and dynamic features.

The advantages of Boost-R are due to three salient features behind this approach: (i) Boost-R leverages the "divide-and-conquer" structure of binary trees to address the inevitable heterogeneity among a large population of individuals; (ii) the non-parametric nature of the algorithm (e.g., binary tree and B-splines) enables us to capture the complex and non-linear relationship between event processes and features, which may not be adequately captured by parametric approaches; (iii) Boost-R is built into the framework of gradient boosted trees, which has proven to be one of the most successful statistical learning approaches over the past decade. Comprehensive numerical studies, including a case study, have been performed to demonstrate the advantages of Boost-R. R code has been made available on GitHub to facilitate the adoption of this new approach.

References

Anderson, P. K., Borgan, O., Gill, R. D., and Keiding, N. (1993), *Statistical Models based on Counting Processes*, New York, NY: Springer-Verlag.

Bacchetti, P. and Segal, M. (1995), "Survival Trees with Time-Dependent Covariates: Application to Estimating Changes in the Incubation Period of AIDS," *Lifetime Data Analysis*, 1, 35–47.

Bou-Hamad, I. (2011), "A Review of Survival Trees," *Statistics Surveys*, 5, 44–71.

Bou-Hamad, I., Larocque, D., Ben-Ameur, H., Masse, L., Vitaro, F., and Tremblay, R. (2009), "Discrete-Time Survial Trees," *Canadian Journal of Statistics*, 37, 17–32.

Chen, T. and Guestrin, C. (2016), "XGBoost: A Scalable Tree Boosting System," *arXiv:*, 1603.02754v3.

Chipman, H. A., George, E. I., and McCulloch, R. E. (2010), "BART: Bayesian Additive Regression Trees," *The Annals of Applied Statistics*, 4, 266–298.

Fan, J., Nunn, M., and Su, X. G. (2009), "Multivariate Exponential Survival Trees and Their Application to Tooth Prognosis," *Computational Statistics and Data Analysis*, 53, 1110–1121.

Fan, J., Su, X. G., Levine, R., Nunn, M., and Leblanc, M. (2006), "Trees for Censored Survival Data by Goodness of Split, with Application to Tooth Prognosis," *Journal of the American Statistical Association*, 101, 959–967.

Fleming, T. R. (1991), *Counting Processes and Survival Analysis*, New York, NY: John Wiley & Sons.

Freund, Y. and Schapire, R. E. (1997), "A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, 55, 119–139.

Grob, G. L., Cardoso, A., Liu, C., Little, D. A., and Chamberlain, B. P. (2018), "A recurrent neural network survival model: predicting web user return time," in *Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD 2018)*, pp. 152–168.

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

Harrell, F., Califf, R., Pryor, D., Lee, K., and Rosati, R. (1982), "Evaluating the Yield of Medical Tests," *Journal of American Medicine Association*, 247, 2543–2546.

Hastie, T., Tibshirani, R., and Friedman, J. (2009), *The Elements of Statistical Learning, 2nd Edition*, New York: Springer.

Hothorn, T., Bühlmann, P., Dudoit, S., Molinaro, A., and Van Der Laan, M. (2006), "Survival Ensembles," *Biostatistics*, 7, 355–373.

Huo, X. M., Kim, S. B., Tsui, K. L., and C., W. S. (2006), "FBP: A Frontier-Based Tree-Pruning Algorithm," *INFORMS Journal on Computing*, 18, 494–505.

Ishwaran, H. and Kogalur, U. B. (2010), "Consistency of Random Survival Forests," *Statistics Probability Letter*, 80, 1056–1064.

Ishwaran, H., Kogalur, U. B., Blackstone, E. H., and Lauer, M. S. (2008), "Random Survival Forests," *The Annals of Applied Statistics*, 2, 841–860.

Ishwaran, H., Kogalur, U. B., Gorodeski, E. Z., Minn, A. J., and Lauer, M. S. (2010), "High-Dimensional Variable Selection for Survival Data," *Journal of the American Statistical Association*, 105, 205–217.

Jordan, M. (2019), "Artificial Intelligence–The Revolution Hasn't Happened Yet," *Harvard Data Science Review*, 1, `https://hdsr.mitpress.mit.edu/pub/wot7mkc1`.

Joseph, V. R. (2016), "Space-filling designs for computer experiments: A review," *Quality Engineering*, 28, 28–35.

Joseph, V. R., Gul, E., and Ba, S. (2015), "Maximum Projection Designs for Computer Experiments," *Biometrika*, 102, 371–380.

Katzman, J. L., Shaham, U., Cloninger, A., Bates, J., Jiang, T., and Kluger, Y. (2018), "Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network," *BMC medical research methodology*, 18:24.

Kelly, P. J. and Lim, L. (2000), "Survival Analysis for Recurrent Event Data: An Application to Childhood Infectious Diseases," *Statistics in Medicine*, 19, 13–33.

Lao, J., Chen, Y., Li, Z.-C., Li, Q., Zhang, J., Liu, J., and Zhai, G. (2017), "A deep learning-based radiomics model for prediction of survival in glioblastoma multiform," *Scientific Report*, 7, 10353.

Lee, C., Zame, W. R., Yoon, J., and van der Schaar, M. (2018), "A deep learning approach to survival analysis with competing risks," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-18)*.

Lewis, P. and Shedler, G. (1979), "Simulation of Nonhomogenous Poisson Processes by Thinning," *Naval Research Logistics Quarterly*, 26, 403–413.

Liu, X. and Pan, R. (2020), "Analysis of Large Heterogeneous Repairable System Reliability Data with Static System Attributes and Dynamic Sensor Measurement in Big Data Environment," *Technometrics*, 62, 206–222.

Meeker, W. Q. and Escobar, L. A. (1998), *Statistical Methods for Reliability Data*, New York, NY: John Wiley & Sons.

Nelson, W. (1995), "Confidence Limits for Recurrence Data: Applied to Cost or Number of Product Repairs," *Technometrics*, 37, 147–157.

Nilsson, R., Pena, J. M., Bjorkegren, J., and Tegner, J. (2007), "Consistent Feature Selection for Pattern Recognition in Polynomial Time," *Journal of Machine Learning Research*, 8, 589–612.

Paynabar, K., Jin, J., and Reed, M. P. (2015), "Informative Sensor and Feature Selection via Hierarchical Nonnegative Garrote," *Technometrics*, 57, 514–523.

Ranganath, R., Perotte, A., Elhadad, N., and Blei, D. (2016), "Deep survival analysis," *Machine Learning for Healthcare Conference*, 101–114.

Reunanen, J. (2003), "Overfitting in Making Comparisons Between Variable Selection Methods," *Journal of Machine Learning Research*, 3, 1371–1382.

Wang, P., Li, Y., and Reddy, C. K. (2017), "Machine Learning for Survival Analysis: A Survey," *arXiv:*, 1708.04649.

Witten, D. M. and Tibshirani, R. (2010), "A Framework for Feature Selection in Clustering," *Journal of the American Statistical Association*, 105, 713–726.

Yuan, M. and Lin, Y. (2005), "Efficient Empirical Bayes Variable Selection and Estimation in Linear Models," *Journal of the American Statistical Association*, 100, 1215–1225.

— (2007), "Model selection and estimation in regression with grouped variables," *Journal of the Royal Statistical Society, Series B*, 68, 49–67.

Appendix A: Discussions on the Ad-Hoc Use of XGBoost for Recurrence Data

[*Appendix A can be moved to supplementary materials if necessary]

If time t is treated as an additional feature, XGBoost is sometimes used in industry to model recurrent event data. Although the off-the-shelf XGBoost can learn the relationship between the cumulative number of events and time and other features, such an *ad hoc* use of XGBoost may have a few major limitations:

▷ When time is treated as another feature, the predicted cumulative intensity is typically "bumpy" because the cumulative events at different times are learned independently. To illustrate this point, Figure 16 below shows how the predicted cumulative intensity from XGBoost and the proposed Boost-R typically look like. Because XGBoost models the number of cumulative failures at different times independently, the predicted intensity at a given time depends on the number of observations available at that time and the observed cumulative events from those available samples at that time. As a result, it is not surprising that the predicted cumulative intensity from XGBoost is rarely smooth. Boost-R, on the other hand, learns a smooth time-dependent function at each tree leaf, and the ensemble prediction of the cumulative intensity is also smooth.

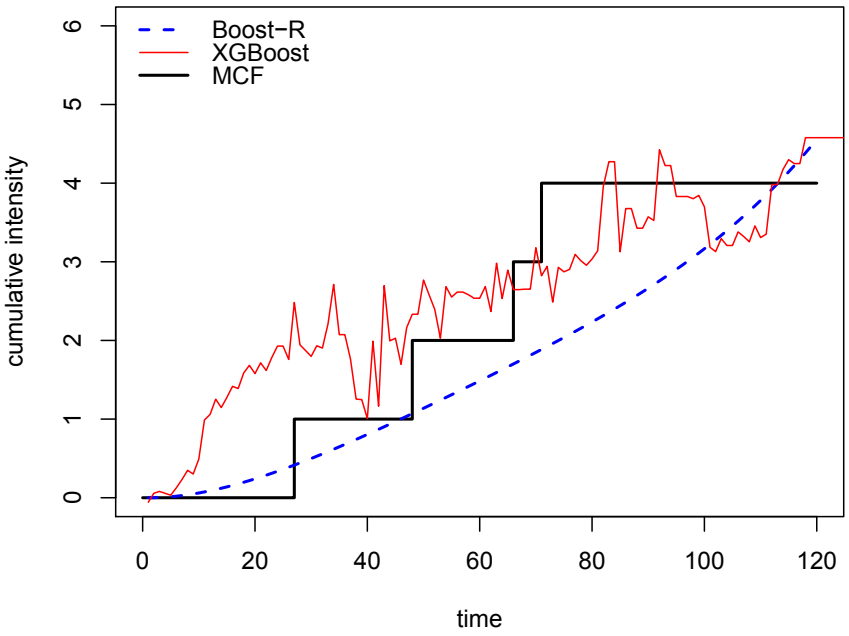

Figure 16 **An example that shows how the predicted cumulative intensity from XGBoost and the proposed Boost-R typically look like. The output from XGBoost appears to be bumpy because it models the cumulative events at different times independently.**

▷ More importantly, if the training data is censored in time (say, at time t_c), the *ad hoc* use of XGBoost (which treats time t as another feature) is incapable of event predictions at a time beyond t_c (i.e., extrapolation in time). This is due to the non-parametric nature of tree-based methods which are ineffective in making extrapolations outside the range of the feature in the training dataset. As a result, although XGBoost can learn the relationship between the cumulative intensity and time (treated as a feature) over the interval $[0, t_c]$, the tree-based method cannot predict the future event process over a time interval beyond the censoring time t_c (i.e., outside the range of "time" in the training dataset).

As clearly shown in the Figure 17 below (more details are provided in the Appendix of the revised manuscript), the predicted cumulative intensity remains a constant if XGBoost is used with time being treated as a feature. In other words, such a use of XGBoost (after treating time t as another feature) is incapable of making predictions beyond the censoring t_c. In the proposed Boost-R, however, because each terminal node contains a time-dependent function (rather than a constant), the predicted cumulative intensity function from Boost-R is smooth and we are able to make extrapolations in time.

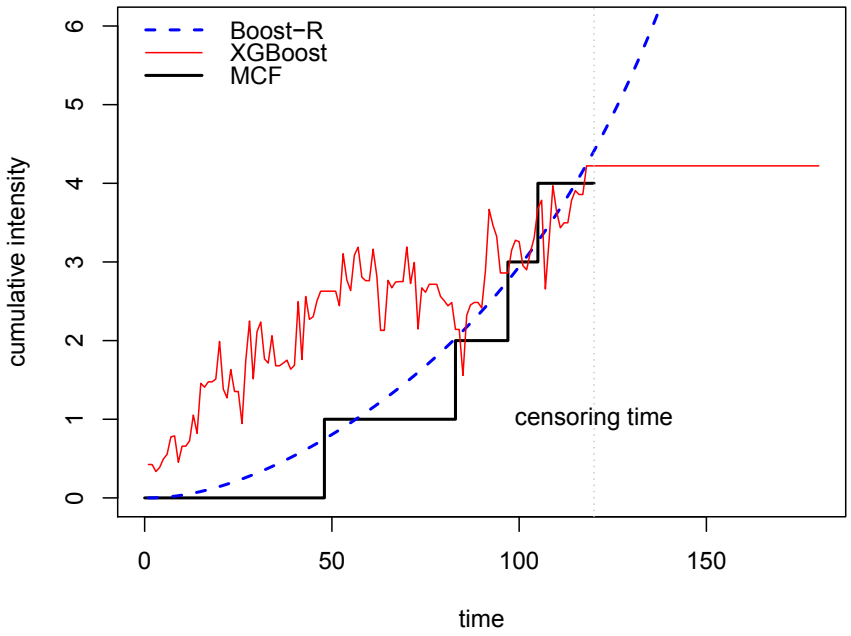

Figure 17 **The ad hoc use of XGBoost is not capable of predicting the event process at a future time interval beyond the censoring time (i.e., extrapolation in time).**

Similarly, if the training data is under Type-II censoring, say, the observation of the process is stopped after observing n_c number of events, the *ad-hoc* use of XGBoost cannot generate predictions that go beyond n_c. For example, after training the XGBoost with a training data under Type-II censoring (censored at n_c), the model cannot answer common questions such as when the $(n_c + 1)$th failure will occur. This is because no sample in the training dataset has more than n_c failures under Type-II censoring, and the non-parametric nature of the conventional tree-based methods prevents the algorithm from making predictions beyond the largest number of failures in the training dataset.

Appendix B: Additional Insights on Boost-R and Comparison Studies

[*Appendix B can be moved to supplementary materials if necessary]

We provide additional discussions and comparison studies of the proposed Boost-R using another application example.

Application. In Kelly and Lim (2000), the authors investigated the recurrent event data modeling for childhood infectious acute respiratory illness (ARI). The goal of the study was to understand the effect of childhood MORbidity from supplementation of VITamin A—the MORVITA trial. This was a randomized double-blinded placebo-controlled trial with 1405 subjects aged 6-47 months. Once a child was randomized (to vitamin A or placebo) they received the same treatment throughout the study.

Each subject has a maximum of four events (i.e., Type-II censoring), and the events are censored if the additive total time since the start of the study is greater than 120 days (i.e., Type-I censoring).

Model. Kelly and Lim (2000) considered a random-effect model. For a subject i, let λ_{ik} be the intensity between the $(k-1)$th event and the kth event $(k=1,2,3,4)$, and

$$\log \lambda_{ik}(t) = \beta_0 + \beta_k Z_{ik} + v_i \tag{22}$$

where β_0 and β_k are the effects, $Z_{ik}=1$ if the subject receives treatment after the $(k-1)$th event otherwise $Z_{ik}=0$, and $v_i \sim N(0,\sigma^2)$ is a random effect covariate that introduces the within-subject correlation.

Different subjects respond to the treatment differently. For example, if the treatment is constantly effective, $\beta_1=\beta_2=\beta_3=\beta_4=-1$. If the treatment is only effective for the first event, $\beta_1=-1$ and $\beta_2=\beta_3=\beta_4=0$.

Data. In our experiment, we simulate the data for 1000 subjects based on the model in Kelly and Lim (2000). For each subjects, we randomly simulate two features x_1 and x_2 from a uniform distribution on $[0,1]$, and consider four potential sub-populations as follows

- if $x_{i,1}, x_{i,2} < 0.5$, the treatment is only effective for the first event, i.e., $\beta_1=-1$ and $\beta_2=\beta_3=\beta_4=0$;
- if $x_{i,1} < 0.5$ and $x_{i,2} \geq 0.5$, the treatment is effective for the first two events, i.e., $\beta_1=\beta_2=-1$ and $\beta_3=\beta_4=0$;
- if $x_{i,1} \geq 0.5$ and $x_{i,2} < 0.5$, the treatment is effective for the first three events, i.e., $\beta_1=\beta_2=\beta_3=-1$ and $\beta_4=0$.
- if $x_{i,1} \geq 0.5$ and $x_{i,2} \geq 0.5$, the treatment is effective for the all four events, i.e., $\beta_1=\beta_2=\beta_3=\beta_4=-1$.

Hence, different subjects respond to treatment differently. Even for subjects from the same sub-group, the random effect, $v_i \sim N(0,\sigma^2)$, further introduces the within-sample correlation. In the subsequent comparison studies, we consider three different values for σ, i.e., 0, 0.1 and 0.4, which were also considered in Kelly and Lim (2000).

All datasets are available on GitHub (`https://github.com/dnncode/Boost-R`).

Comparison. Three methods are included in the comparison study: Boost-R, RF-R and XGBoost (described in Appendix A). For each method, data from 500 subjects are used to train the model, and data from the remaining 500 subjects are used to test the model performance. Figures 18, 19 and 20 below show the box plot of the squared L^2 distance between the predicted cumulative intensity and the observed cumulative intensity

over the time interval from 0 to 120 days, respectively for three datasets assuming different values for σ, i.e., 0, 0.1 and 0.4.

In each figure, four combinations of the tuning parameters (γ_1 and γ_2) are used for Boost-R models. In particular, Boost-R-1, Boost-R-2, Boost-R-3 and Boost-4 are respectively based on the following combinations of γ_1 and γ_2: $(10, 10)$, $(10, 50)$, $(50, 10)$ and $(50, 50)$. Three different choices of the learning rate, including 0.1, 0.5 and 1, are respectively used for XGBoost-1, XGBoost-2 and XGBoost-3. RF-F does not involve major tuning parameters.

Figures 18, 19 and 20 all indicate that the proposed Boost-R provides the best performance for all three datasets. In fact, considering the additional limitations of the ad-hoc use of XGBoost (discussed in Appendix A), Boost-R appears to be a good choice for such an application. In addition, the performance of all methods deteriorates when σ becomes larger, as expected.

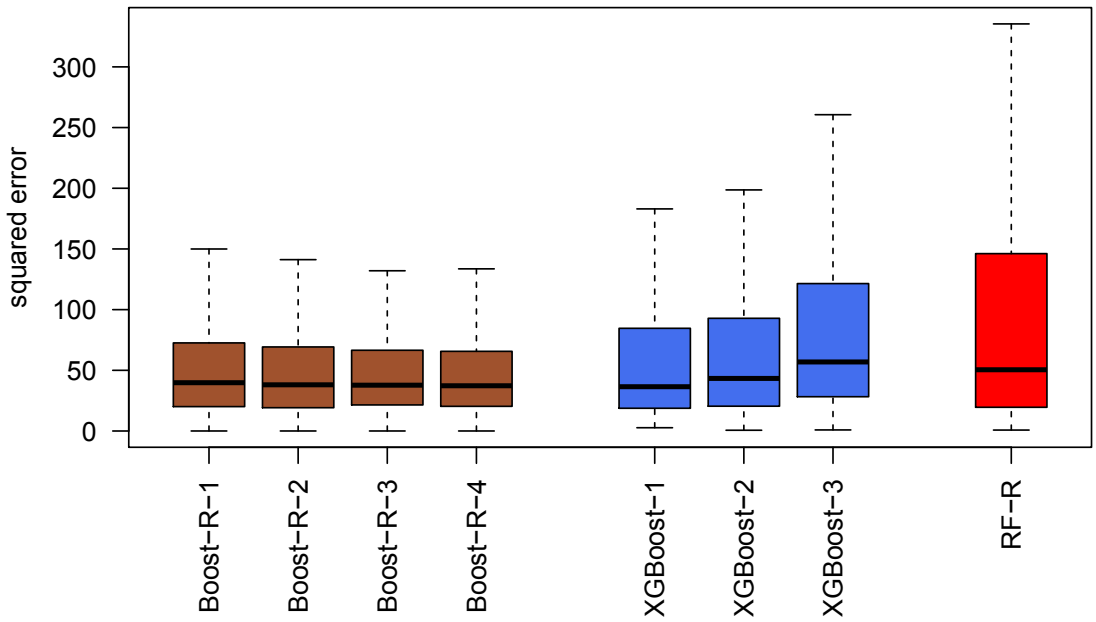

Figure 18 **Comparison between Boost-R, RF-R and XGBoost: box plot of the squared L^2 distance between the predicted cumulative intensity and the observed cumulative intensity over 0 to 120 days ($\sigma = 0$)**

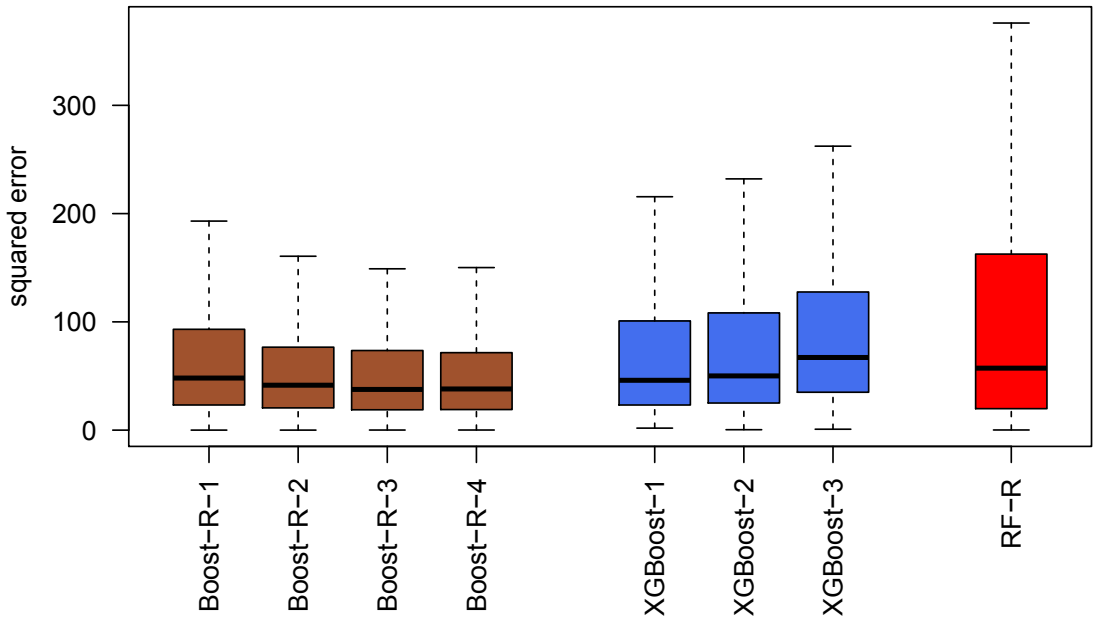

Figure 19 **Comparison between Boost-R, RF-R and XGBoost: box plot of the squared L^2 distance between the predicted cumulative intensity and the observed cumulative intensity over 0 to 120 days ($\sigma = 0.1$)**

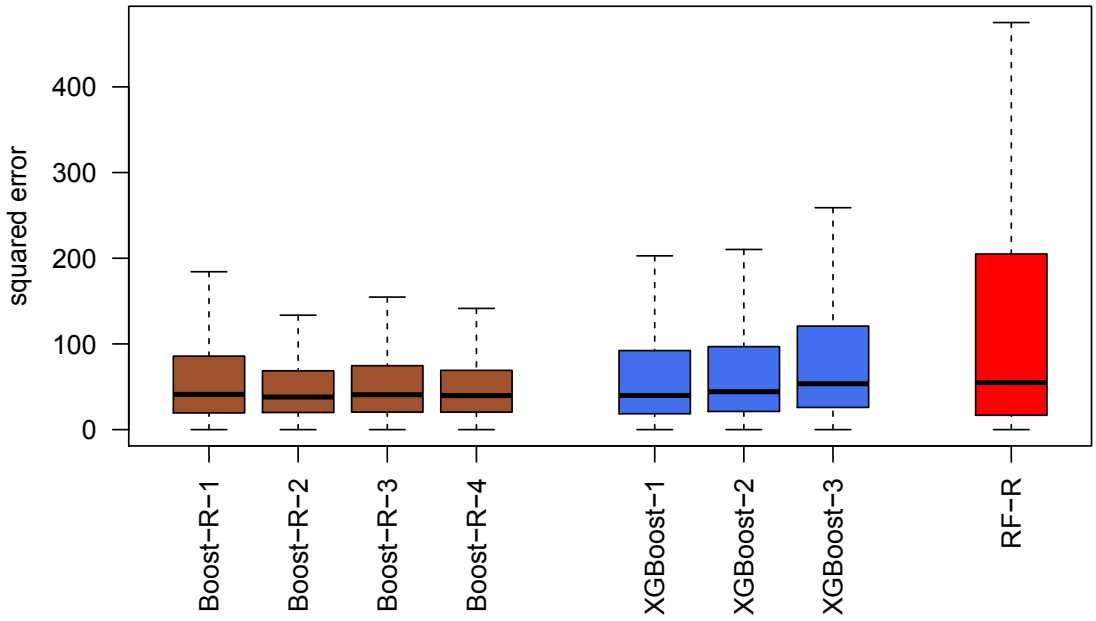

Figure 20 **Comparison between Boost-R, RF-R and XGBoost: box plot of the squared L^2 distance between the predicted cumulative intensity and the observed cumulative intensity over 0 to 120 days ($\sigma = 0.4$)**

We further compare the extrapolation capabilities between Boost-R and the ad-hoc use of XGBoost. In particular, we train the model using the data from 500 subjects over the time interval between 0 and 120 days, and use the model to predict the cumulative number of events at 240 days for the 500 subjects in the testing dataset (i.e., extrapolation in time). Figures 21, 22 and 23 shows the box plot of the MSE of the predicted cumulative number of events at 240 days. All three figures well illustrate the advantage of Boost-R in terms of extrapolating the number of events beyond the censoring time. Of course, as already discussed in Appendix A, the non-parametric nature of tree-based methods prevents the ad-hoc use of XGBoost to predict the future event process beyond the censoring time (i.e., outside the range of "time" in the training dataset).

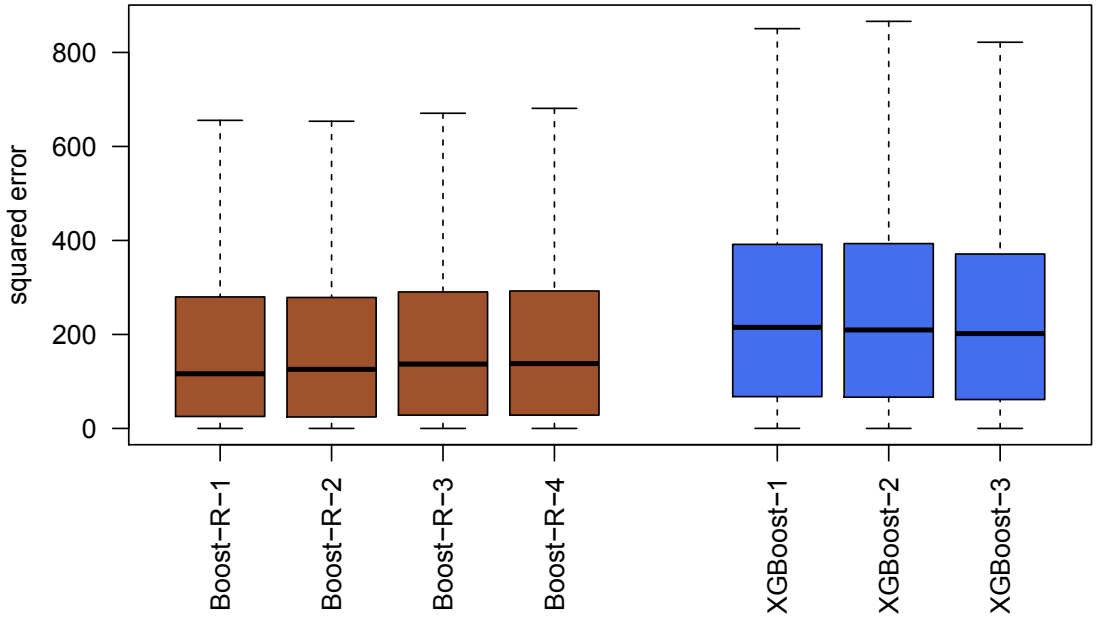

Figure 21 **Box plot of the MSE of the predicted cumulative number of events at 240 days ($\sigma = 0$)**

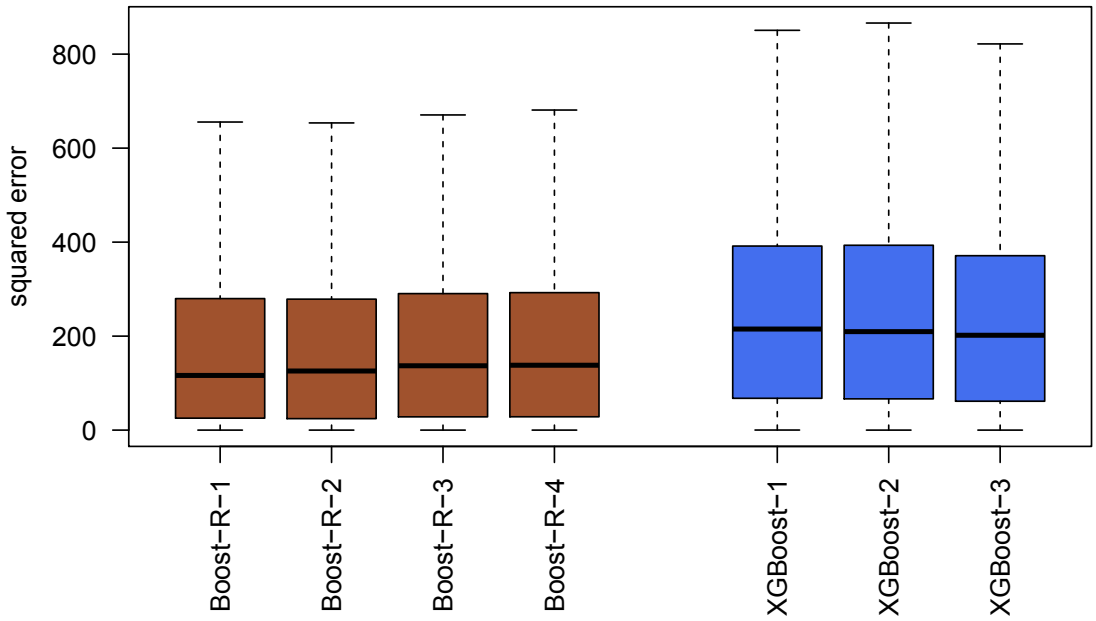

Figure 22 **Box plot of the MSE of the predicted cumulative number of events at 240 days ($\sigma = 0.1$)**

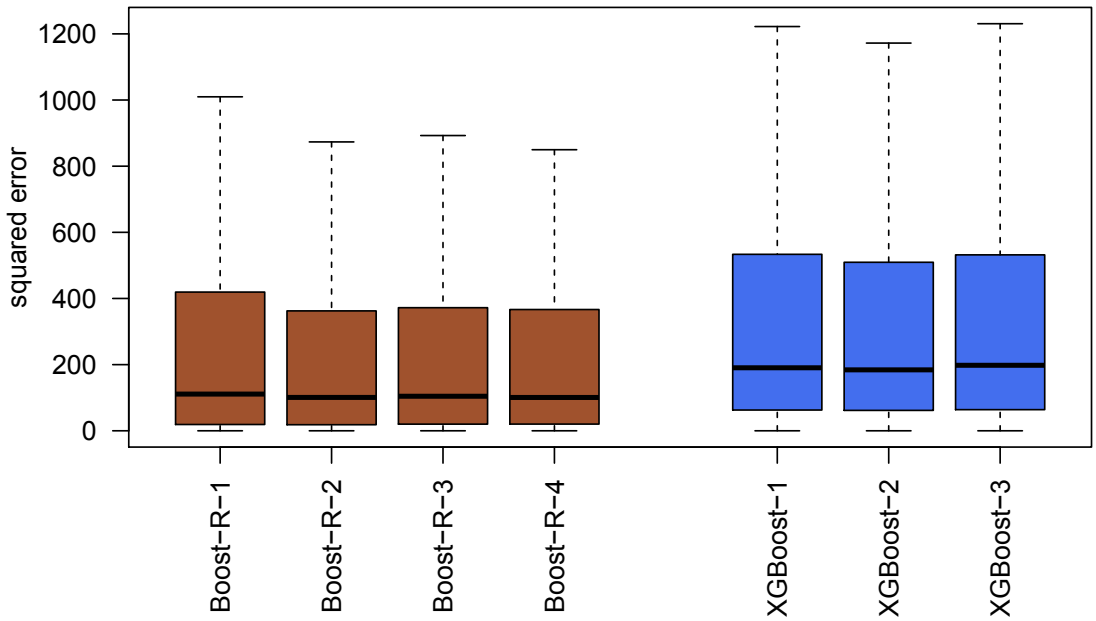

Figure 23 **Box plot of the MSE of the predicted cumulative number of events at 240 days ($\sigma = 0.4$)**