Theory and software for boosted nonparametric hazard estimation

Donald K. K. Lee DONALD.LEE@EMORY.EDU
Goizueta Business School and Department of Biostatistics & Bioinformatics, Emory University

Ningyuan Chen NINGYUAN.CHEN@UTORONTO.CA
Rotman School of Management, University of Toronto

Hemant Ishwaran HEMANT.ISHWARAN@GMAIL.COM
Division of Biostatistics, University of Miami

Xiaochen Wang XCWANG11@GMAIL.COM and **Hongyu Zhao** HONGYU.ZHAO@YALE.EDU
Department of Biostatistics, Yale University

Arash Pakbin A.PAKBIN@TAMU.EDU and **Bobak J. Mortazavi** BOBAKM@TAMU.EDU
Department of Computer Science & Engineering, Texas A&M University

Abstract

Nonparametric approaches for analyzing survival data in the presence of time-dependent covariates is a timely topic, given the availability of high frequency data capture systems in healthcare and beyond. We present a theoretically justified gradient boosted hazard estimator for this setting, and describe a tree-based implementation called `BoXHED` (pronounced 'box-head') that is available from GitHub: `www.github.com/BoXHED`. Our numerical study demonstrates that there is a place in the machine learning toolbox for a nonparametric method like `BoXHED` that can flexibly handle time-dependent covariates. The results presented here are distilled from the recent works of Lee et al. (2021) and Wang et al. (2020).

Keywords: Survival Analysis, Gradient Boosting, Functional Data, Nonparametric Likelihood

1. Introduction

Time-dependent covariates are becoming increasingly common in modern applications of survival analysis. For example in medicine, electronic health records systems make it possible to log patient vitals throughout the day, and these measurements can be used to build real-time warning systems for adverse outcomes such as in-ICU mortality. Another example comes from financial technology, where lenders track obligors' behaviours over time to assess and revise default rate estimates. The ability to flexibly model the complex interactions among the time-varying risk factors is crucial to prognosticating risk accurately in real-time, be it mortality risk or default risk. There is thus a need for survival methods that:

i) Can incorporate potentially high-dimensional covariates that evolve in continuous time; and

ii) Are nonparametric, i.e. do not require parametric assumptions (e.g. Weibull) or semiparametric ones (e.g. proportional hazards); and

iii) Preferably have a sound theoretical basis.

However, much of the recent works in the nonparametric survival literature in machine learning focus on the special case of time-static covariates $X = (X^{(1)}, \cdots, X^{(p)})$ (Ishwaran et al., 2008; Ranganath et al., 2016; Bellot and van der Schaar, 2018, 2019; Lee et al., 2019). For this setting, a popular approach is to estimate the conditional survivor function for the event time T,

$$S(t|X) = \mathbb{P}(T > t|X) = \exp\left(-\int_0^t \lambda(u, X)du\right), \tag{1}$$

where $\lambda(t, x)$ is the hazard function.

In the general setting where the covariates $X(t) = (X^{(1)}(t), \cdots, X^{(p)}(t))$ are time-varying, there is no meaningful analogue to the survivor function (1), since it involves integrating $\lambda(u, X(u))$ along the unknown future trajectory of the covariates $\{X(u)\}_{u\in(0,t]}$. Instead of the survivor function, the interest is now on estimating the hazard $\lambda(t, x)$ given $X(t) = x$, which is informally the probability of the event happening in the near future given it has not happened yet:

$$\mathbb{P}(T \in [t, t + dt)|T \geq t, X(t) = x) \approx \lambda(t, x)dt. \tag{2}$$

Note that in the time-static covariate setting, we can recover $S(t|x)$ from $\lambda(t, x)$ via (1). It follows that the hazard is the fundamental quantity that unifies the analyses of both settings. Recognizing its importance, recent works in the machine learning literature have proposed neural network models for the hazard function in the discrete-time setting (Jarrett et al., 2018; Ren et al., 2019). This is equivalent to solving a series of binary classification problems, one at each point in an equally spaced time-grid.

To extend hazard modelling to the continuous time setting, we propose a hazard estimator using gradient boosting that satisfies i) to iii). The estimator can be used with any weak learners including regression trees, and it consistently recovers the true hazard under mild identifiability conditions. Furthermore, we describe a novel software implementation of the estimator called `BoXHED` (Boosted eXact Hazard Estimator with Dynamic covariates) that is available from GitHub: `www.github.com/BoXHED`. The comparative performance of `BoXHED` will be illustrated using simulated data.

2. Theory overview

Nonparametric hazard estimation via gradient boosting is historically a challenging problem, especially in the time-dependent covariate setting. Most of the existing boosting approaches focus on the Cox proportional hazards model for time-static covariates, which apply gradient boosting to the Cox partial likelihood loss (Ridgeway, 1999; Li and Luan, 2005; Bühlmann and Hothorn, 2007; Binder and Schumacher, 2008). An alternative approach uses flexible transformation models of parametric families to allow for time-dependent covariates but with time-static effects (Hothorn, 2019). To date there is no fully nonparametric boosting solution for handling time-dependent covariates. This is because the nonparametric log-likelihood seemingly does not have a gradient, unlike standard applications of gradient boosting where the gradient can easily be identified and calculated.

To explain why, let us first fix the survival setting, which comes from the seminal work of Aalen (1978): In addition to the quantities appearing in (2), we now also have an

indicator $Y(t) \in \{0,1\}$ denoting whether the subject is at risk of experiencing the event during $[t, t+dt)$. Both $X(t)$ and $Y(t)$ are assumed to be predictable processes. Under this setting, the probability (2) of experiencing the event during $[t, t+dt)$ generalizes to $\lambda(t,x)Y(t)dt$. This allows for a variety of censoring schemes including right-censoring, where $Y(t)$ is nonincreasing. Without loss of generality let us also normalize the units of time so that $Y(t) = 0$ for $t > 1$, i.e. the subject is not a risk after time $t = 1$. This allows us to restrict attention to the time interval $(0, 1]$.

If the subject experiences the event in $(0, 1]$ then $\Delta = Y(T)$ equals 1, otherwise $\Delta = 0$ and we set T to an arbitrary number greater than 1. Given n functional data samples $\{X_i(\cdot), Y_i(\cdot), T_i\}_{i=1}^n$, the evolution of subject i's event status can then be thought of as a sequence of coin flips at time increments $t = 0, dt, 2dt, \cdots$, with the probability of "heads" at each time point equal to $\lambda(t,x)Y(t)dt$. Therefore, subject i's contribution to the likelihood is

$$\{1 - \lambda(0, X_i(0))Y_i(0)dt\} \times \{1 - \lambda(dt, X_i(dt))Y_i(dt)dt\} \times \cdots \times \lambda(T_i, X_i(T_i))^{\Delta_i}$$

$$\xrightarrow[dt \downarrow 0]{} e^{-\int_0^1 Y_i(t)\lambda(t, X_i(t))dt} \lambda(T_i, X_i(T_i))^{\Delta_i},$$

where the limit can be understood as a product integral. Hence, if the log-hazard function is $F(t,x) = \log \lambda(t,x)$ then the (scaled) negative log-likelihood functional is

$$R_n(F) = \frac{1}{n}\sum_{i=1}^n \int_0^1 Y_i(t) e^{F(t, X_i(t))} dt - \frac{1}{n}\sum_{i=1}^n \Delta_i F(T_i, X_i(T_i)), \tag{3}$$

which we shall refer to as the likelihood risk. The goal is to estimate the hazard function $\lambda(t,x) = e^{F(t,x)}$ nonparametrically by minimizing $R_n(F)$.

However, the chief difficulty with minimizing $R_n(F)$ using gradient boosting is that the canonical representation of the likelihood risk does not have a gradient. To see this, observe that the directional derivative of (3) equals

$$\frac{d}{d\theta} R_n(F + \theta f)\Big|_{\theta=0} = \frac{1}{n}\sum_{i=1}^n \int_0^1 Y_i(t) e^{F(t, X_i(t))} f(t, X_i(t)) dt - \frac{1}{n}\sum_{i=1}^n \Delta_i f(T_i, X_i(T_i)),$$

which is the difference of two different inner products $\langle e^F, f\rangle_\dagger - \langle 1, f\rangle_\ddagger$ where $\langle g, f\rangle_\dagger = \frac{1}{n}\sum_{i=1}^n \int_0^1 Y_i(t) g(t, X_i(t)) f(t, X_i(t)) dt$ and $\langle g, f\rangle_\ddagger = \frac{1}{n}\sum_{i=1}^n \Delta_i g(T_i, X_i(T_i)) f(T_i, X_i(T_i))$. Hence, the directional derivative cannot be expressed as a single inner product of the form $\langle g_F, f\rangle$ for some function $g_F(t,x)$. Were it possible to do so, g_F would then be the gradient function.

In simpler non-functional data settings like regression or classification, the loss can be written as $L(Y, F(x))$, where F is the non-functional statistical target and Y is the outcome, so the gradient is simply $\partial L(Y, F(x))/\partial F(x)$. The negative gradient is then approximated by a base learner from a predefined function class $\mathcal{F}$ (e.g. linear functions or tree learners), and the approximation is used as the direction of update in a boosting iteration. Importantly, in the simpler non-functional data setting the gradient does not depend on the space that F belongs to.

By contrast, our key insight that resolves the challenge with nonparametric hazard boosting is that the gradient of $R_n(F)$ can only be defined after carefully specifying an appropriate sample-dependent domain for $R_n(F)$. The likelihood risk can then be re-expressed as a smooth convex functional, and an analogous representation also exists for the population risk. These representations not only allow us to describe and implement a gradient boosting procedure, but are also crucial to establishing the consistency of our estimator. In the interest of space we will specialize the general representation for $R_n(F)$ to the case where the base learner class consists of regression trees, which is what we use to implement our estimator. The general treatment of arbitrary learner classes can be found in Lee et al. (2021).

Since a tree learner $F(t,x)$ in the hazard setting is a piecewise constant function of both t and $x = (x^{(1)}, \cdots, x^{(p)})$, we can express it as a weighted sum of indicator functions for disjoint time-covariate regions $\{B_j\}_j$, i.e. $F \in \mathcal{F} = \left\{\sum_j c_j I_{B_j}(t,x) : c_j \in \mathbb{R}\right\}$. The most common way for generating these regions is to use axis parallel cuts when growing the tree, in which case the regions are hypercubes of the form

$$B = \left\{ (t,x) \quad : \quad \begin{array}{c} \underline{t}^B < t \le \overline{t}^B \\ \underline{x}^{(1,B)} < x^{(1)} \le \overline{x}^{(1,B)} \\ \vdots \\ \underline{x}^{(p,B)} < x^{(p)} \le \overline{x}^{(p,B)} \end{array} \right\}. \tag{4}$$

It can then be verified that the likelihood risk (3) and its gradient function evaluated at F are respectively

$$R_n(F) = \sum_{j:\mu_j>0} \left(e^{c_j}\mu_j - \frac{c_j \upsilon_j}{n} \right), \qquad g_F(t,x) = \sum_{j:\mu_j>0} \left(e^{c_j} - \frac{\upsilon_j}{n\mu_j} \right) I_{B_j}(t,x), \tag{5}$$

where

$$\mu_j = \frac{1}{n}\sum_{i=1}^n \int_0^1 Y_i(t) \cdot I[t, X_i(t) \in B_j], \quad \upsilon_j = \frac{1}{n}\sum_{i=1}^n \Delta_i I[\{T_i, X_i(T_i)\} \in B_j].$$

Note that υ_j is the (scaled) number of events observed in the time-covariate region B_j, while μ_j measures the denseness of the sample trajectories inside B_j. We can use $\{\mu_j\}_j$ to define the empirical correlation between two tree learners $F_1(t,x) = \sum_j c_{1j} I_{B_j}(t,x)$ and $F_2(t,x) = \sum_j c_{2j} I_{B_j}(t,x)$ as

$$\langle F_1, F_2 \rangle_n = \frac{\sum_j \mu_j c_{1j} c_{2j}}{\sqrt{\sum_j \mu_j c_{1j}^2} \cdot \sqrt{\sum_j \mu_j c_{2j}^2}}. \tag{6}$$

Positive values of $\langle F_1, F_2 \rangle_n$ then imply that F_1 and F_2 are aligned, and the closer $\langle F_1, F_2 \rangle_n$ is to 1 the closer the alignment is. Note that even if F_1 and F_2 are not defined on the same set of regions $\{B_j\}_j$, we can always find a finer partition $\{B'_j\}_j$ such that $F_1(t,x) = \sum_j c'_{1j} I_{B'_j}(t,x)$ and $F_2(t,x) = \sum_j c'_{2j} I_{B'_j}(t,x)$.

Algorithm 1 presents the procedure for performing boosted nonparametric hazard estimation using generic tree learners. In essence the algorithm creates a sequence of tree learners $\mathfrak{g}_0(t,x), \mathfrak{g}_1(t,x), \cdots, \mathfrak{g}_{M-1}(t,x)$ in an iterative manner, and uses them to form an ensemble estimator of the log-hazard function $F_M(t,x) = F_0 - \nu \sum_{m=0}^{M-1} \mathfrak{g}_m(t,x)$, from which the hazard estimator can be obtained as $\hat{\lambda}_{\text{boost}}(t,x) = e^{F_M(t,x)}$. Here, M is the number of boosting iterations and $\nu \ll 1$ is the learning rate. The initial guess for the log-hazard, $F_0 = \log(\sum_i \Delta_i / \sum_j \mu_j)$, is the best constant that minimizes (3).

At the m-th boosting iteration, the tree learner $\mathfrak{g}_m$ is obtained by approximating the gradient g_{F_m} with a shallow regression tree. The degree of approximation is measured by $\langle \mathfrak{g}_m, g_{F_m} \rangle_n$, which improves with more tree splits, but this comes at the cost of a more complex $\mathfrak{g}_m$ that is prone to overfit. In (7) in Algorithm 1 we allow for the use of any approximation that has a correlation of at least ε with g_{F_m} for a fixed $\varepsilon \in (0,1]$. In practice a small value of ε is desired, which corresponds to simple tree learners with a small number of splits.

Algorithm 1 Boosted nonparametric hazard estimation with generic tree learners

Input #iterations M, learning rate $\nu \ll 1$, correlation $\varepsilon \in (0,1]$.
Initialize $F_0 = \log(\sum_{i=1}^n \Delta_i / \sum_j \mu_j)$.
for $m = 0$ **to** $M-1$ **do**
Compute an approximation $\mathfrak{g}_m$ to the gradient g_{F_m} that satisfies

$$\langle \mathfrak{g}_m, g_{F_m} \rangle_n > \varepsilon. \tag{7}$$

Compute $F_{m+1} \leftarrow F_m - \nu \mathfrak{g}_m$.
Output: $\hat{\lambda}_{\text{boost}}(t,x) = e^{F_M(t,x)}$.
end for

Under mild identifiability conditions and by scaling M and ν in an appropriate manner with sample size n, it can be shown that $\hat{\lambda}_{\text{boost}}$ converges in probability to the best tree-based hazard estimator among $\{e^F : F \in \mathcal{F}\}$. Here, 'best' is defined as the hazard estimator closest to the true hazard λ in the metric induced by $\|F_1 - F_2\| = \sqrt{\mathbb{E}\,\langle F_1 - F_2, F_1 - F_2 \rangle_n}$, where the expectation is with respect to the functional data samples. Formally, $\hat{\lambda}_{\text{boost}}$ satisfies the oracle equality

$$\left\| \hat{\lambda}_{\text{boost}} - \lambda \right\|^2 = \min_{F \in \mathcal{F}} \left\| e^F - \lambda \right\|^2 + o_p(1).$$

See Proposition 4 in Lee et al. (2021) for details.

3. `BoXHED` implementation

Algorithm 1 supports a variety of ways for implementing $\mathfrak{g}_m$, which needs to be sufficiently aligned with the gradient. Since the representation (5) for $R_n(F)$ is convex, $\mathfrak{g}_m$ is aligned with the gradient if and only if $R_n(F_m)$ can be decreased by moving in the direction of $-\mathfrak{g}_m$. Furthermore, the larger the decrease, the greater the alignment. This key insight leads to

an approach for growing the tree $\mathfrak{g}_m$ that is similar to the spirit of `XGBoost` (Chen and Guestrin, 2016), albeit even more targeted at risk reduction: Starting with a tree with a root node, we choose the split that maximally reduces the *exact* likelihood risk, and repeat the process iteratively on successive leaf nodes until the tree has been split L times. The `BoXHED` package (Wang et al., 2020) is an instance of Algorithm 1 that we implement using this approach. The current version (1.0) is written in Python and is specialized to right censored data, i.e. $Y_i(t) = I(t \leq \tilde{T}_i)$ where $\tilde{T}_i$ is the minimum of the event time and the censoring time for subject i. In this case the likelihood risk (3) reduces to

$$R_n(F) = \frac{1}{n}\sum_{i=1}^{n}\left\{\int_0^{\tilde{T}_i} e^{F(t,X_i(t))}dt - \Delta_i F(\tilde{T}_i, X_i(\tilde{T}_i))\right\}.$$

The specific way `BoXHED` constructs $\mathfrak{g}_m$ is as follows. Initialize $\mathfrak{g}_{m,0}(t,x) = 0$ to be the tree with just a root node, and let

$$\mathfrak{g}_{m,l}(t,x) = \sum_{j=1}^{l+1} c_{m,j} I_{B_{m,j}}(t,x)$$

be the intermediate tree after l splits. Here, the splits can be on the covariates or on time. Each of the time-covariate regions $B_{m,1}, \cdots, B_{m,l+1}$ represent one of the leaf nodes in the intermediate tree, and $c_{m,1}, \cdots, c_{m,l+1}$ are the values of the tree function in those leaf nodes. To obtain $\mathfrak{g}_{m,l+1}(t,x)$, we split one of the $B_{m,j}$ regions into two subregions A_1 and A_2 of the same form as (4) to get

$$\mathfrak{g}_{m,l+1}(t,x) = \mathfrak{g}_{m,l}(t,x) - c_{m,j} I_{B_{m,j}}(t,x) + \gamma_1 I_{A_1}(t,x) + \gamma_2 I_{A_2}(t,x).$$

The region $B_{m,j}$ to split, the variable or time axis to split on and the location of the split, and also the values of (γ_1, γ_2) are all chosen to minimize $R_n(F_m - \mathfrak{g}_{m,l+1})$.

Whereas `XGBoost` minimizes a second order Taylor approximation to a risk function in order to speed up computations, `BoXHED` is able to directly minimize the exact likelihood risk in an efficient way: Since the tree values γ_1, γ_2 only apply to the subregions A_1, A_2, we can write $R_n(F_m - \mathfrak{g}_{m,l+1})$ as

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{2}\left\{\int_0^{\tilde{T}_i} e^{F_m(t,X_i(t))-\gamma_k} I_{A_k}(t,X_i(t))dt - \Delta_i\left[F_m(\tilde{T}_i, X_i(\tilde{T}_i)) - \gamma_k\right] I_{A_k}(\tilde{T}_i, X_i(\tilde{T}_i))\right\} + C \\
=&\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{2}\left\{e^{-\gamma_k}\cdot\int_0^{\tilde{T}_i} e^{F_m(t,X_i(t))} I_{A_k}(t,X_i(t))dt + \gamma_k\cdot\Delta_i I_{A_k}(\tilde{T}_i, X_i(\tilde{T}_i))\right\} + C'
\end{aligned}$$

where C, C' are quantities that do not depend on γ_1 or γ_2. Rewriting the above yields

$$R_n(F_m - \mathfrak{g}_{m,l+1}) = \sum_{k=1}^{2}[e^{-\gamma_k}U_k + \gamma_k V_k] + C', \tag{8}$$

where $U_k = n^{-1}\sum_{i=1}^{n}\int_0^{\tilde{T}_i} e^{F_m(t,X_i(t))} I_{A_k}(t,X_i(t))dt$ and $V_k = n^{-1}\sum_{i=1}^{n}\Delta_i I[\{\tilde{T}_i, X_i(\tilde{T}_i)\} \in A_k]$. Note that V_k is the (scaled) number of events observed in A_k. If $U_k, V_k > 0$ then the

minimizing value in A_k is $\gamma_k = \log(U_k/V_k)$. Substituting this into (8) yields the minimized value $R(F_m - \mathfrak{g}_{m,l+1}) = \sum_{k=1}^{2} V_k \left(1 + \log \frac{U_k}{V_k}\right) + C'$. By reasoning inductively, the decrease in the likelihood risk due to the new split is

$$\begin{aligned} d &= R(F_m - \mathfrak{g}_{m,l+1}) - R(F_m - \mathfrak{g}_{m,l}) \\ &= V_1 \left(1 + \log \frac{U_1}{V_1}\right) + V_2 \left(1 + \log \frac{U_2}{V_2}\right) - (V_1 + V_2) \left(1 + \log \frac{U_1 + U_2}{V_1 + V_2}\right), \end{aligned} \tag{9}$$

which can be viewed as a score for determining the best split: Of all possible splits (defined by $B_{m,j}$, A_1, and A_2) where $U_k, V_k > 0$ in both subregions, the best one is that which minimizes (9). Since at each iteration only two new leaf nodes are created, it is only necessary to determine the best split for each of the two new regions at the next iteration: d remains unchanged for the other leaf nodes from previous iterations.

Candidate split points for variables. Given a set of candidate split points for a particular variable, the best split point is that which minimizes (9). If $x^{(j)}$ is continuous, `BoXHED` proposes split candidates based on the percentiles of the observed data for $x^{(j)}$. The default setting places a candidate split at every decile.

If $x^{(j)}$ is categorical, `BoXHED` employs a one-hot encoding heuristic: Set A_1 equal to the intersection of $B_{m,j}$ with the region where $x^{(j)}$ equals a particular categorical label. Hence A_2 is the intersection of $B_{m,j}$ with the region where $x^{(j)}$ is any other label. The algorithm would then choose the category label for A_1 that minimizes the score (9). The rationale for this is that if U_k and V_k can be varied continuously, then (9) tends to $-\infty$ as the ratio U_1/V_1 tends to ∞. Hence a heuristic is to find a subset of categorical labels to intersect with $B_{m,j}$ so that U_1/V_1 is maximized. This always has a solution in the form of a singleton set.

Defining variable importance. A variable importance measure can be constructed for the `BoXHED` estimator: Define the importance of the k-th variable (the zero-th one being time t) as $\mathcal{I}_k = \sum_{m=0}^{M-1} \mathcal{I}_k(\mathfrak{g}_m)$, where for tree $\mathfrak{g}_m$ with L internal nodes $\mathcal{I}_k(\mathfrak{g}_m) = -\sum_{\ell=1}^{L} d_\ell I(v(\ell) = k) \geq 0$. Here, d_ℓ is the split score (9) at iteration ℓ and $v(\ell)$ is the variable used for the partition. Hence the second sum represents the total reduction in likelihood risk due to splits on the k-th variable in the m-th tree, and $\mathcal{I}_k$ is the total risk reduction across all the trees. To convert $\mathcal{I}_k$ into a measure of relative importance between 0 and 1, it is scaled by $\max_k \mathcal{I}_k$, where a larger value confers higher importance.

4. Numerical Study

To illustrate the value that `BoXHED` adds to the survival analysis toolbox, we compare its performance to those of several existing methods. We use simulated datasets for which the true hazard function is known in order to compute how well the methods do in recovering the truth. The performance metric is the L^2-error, which is calculated on test datasets of N randomly sampled data points $\text{err}_{L^2} = \left\{\frac{1}{N}\sum_{i=1}^{N}(\hat{\lambda}_i - \lambda_i)^2\right\}^{1/2}$, where $\hat{\lambda}_i$ and λ_i are the predicted and true hazard values for the i-th test data point. Further comparisons based on the time-dependent AUC (Blanche et al., 2019), as well as a detailed analysis of a cardiovascular disease dataset from the Framingham Heart Study, can be found in Wang et al. (2020).

		Estimator			
		`BoXHED`	`kernel`	`flexsurv`	`blackboost`
λ_1	0	0.17(0.17, 0.17)	**0.14(0.14, 0.15)**	0.53(0.52, 0.54)	0.58(0.57, 0.59)
	20	**0.20(0.20, 0.20)**	3.4(3.0, 3.9)	0.54(0.53, 0.54)	0.58(0.57, 0.59)
	40	**0.21(0.20, 0.21)**	43(5.7, 80)	0.54(0.54, 0.55)	0.58(0.57, 0.59)
λ_2	0	0.23(0.23, 0.24)	**0.11(0.11, 0.12))**	1.1(1.1, 1.1)	1.4(1.4, 1.4)
	20	**0.25(0.25, 0.26)**	4.5(3.9, 5.2)	1.1(1.1, 1.1)	1.4(1.4, 1.4)
	40	**0.26(0.26, 0.27)**	29(11, 46)	1.1(1.1, 1.1)	1.4(1.4, 1.4)
λ_3	0	0.038(0.037, 0.040)	0.046(0.044, 0.049)	**0.0040(0.0039, 0.0041)**	0.10(0.10, 0.11)
	20	0.047(0.046, 0.049)	1.8(1.1, 2.5)	**0.020(0.019, 0.020)**	0.10(0.10, 0.11)
	40	0.050(0.048, 0.051)	7.6(5.3, 9.7)	**0.030(0.029, 0.031)**	0.10(0.10, 0.11)
λ_4	0	0.049(0.048, 0.050)	**0.045(0.044, 0.046)**	0.20(0.19, 0.20)	0.20(0.19, 0.20)
	20	**0.060(0.059, 0.062)**	3.9(0.66, 7.1)	0.20(0.19, 0.20)	0.20(0.19, 0.20)
	40	**0.069(0.067, 0.070)**	5.5(4.3, 6.7)	0.20(0.20, 0.21)	0.20(0.19, 0.20)

Table 1: err_{L^2} with 95% confidence intervals. The hazard function used in each simulation and the number of irrelevant covariates present ($\{0, 20, 40\}$) is provided in the left column. `BoXHED`'s hyper-parameters are tuned to the training data, whereas those for the other methods are tuned directly to the test data. Note that this puts `BoXHED` at an disadvantage. Furthermore, `flexsurv` includes the log-normal distribution as one of its parametric options, so it is correctly specified for λ_3.

Baseline Comparisons. We compare `BoXHED` to several existing hazard estimation methods: Kernel smoothing (Nielsen and Linton, 1995), a nonparametric hazard estimator for low dimensional covariate settings; parametric hazard estimators for time-dependent covariates (`flexsurv` in R); and boosted parametric estimators for time-static covariates (`blackboost` in R). The Cox proportional hazards model is excluded because it is only able to estimate the cumulative hazard but not the hazard itself. The hyper-parameters[1] for `BoXHED` are tuned using five-fold cross-validation on the training data. For kernel smoothing we utilize the kernel function $K(u) = \frac{3003}{2048}(1-x^2)^6 I(-1 < x < 1)$ from Pérez et al. (2013). The hyper-parameters[2] for the baseline estimators are tuned directly to the test data. Note that this puts `BoXHED` at a significant disadvantage.

Simulated datasets. We simulate four datasets from the following hazard functions used in Pérez et al. (2013), with x_t being a piecewise-constant function with values drawn from $U(0, 1]$:

$$\lambda_1(t, x_t) = B(t, 2, 2) \times B(x_t, 2, 2), \quad \lambda_2(t, x_t) = B(t, 4, 4) \times B(x_t, 4, 4), \quad t \in (0, 1],$$
$$\lambda_3(t, x_t) = \frac{1}{t}\frac{\phi(\log t - x_t)}{\Phi(x_t - \log t)}, \quad \lambda_4(t, x_t) = \frac{3}{2}t^{\frac{1}{2}} \exp\left(-\frac{1}{2}\cos(2\pi x_t) - \frac{3}{2}\right), \quad t \in (0, 5],$$

where $B(\cdot, a, a)$ is the PDF of the Beta distribution with shape and scale parameters equal to a, and $\phi(\cdot)$ and $\Phi(\cdot)$ denote the PDF and CDF of $N(0, 1)$. In other words, λ_1 and λ_2

1. Candidates $L \in \{1, 2, 3, 4\}$ and $M \in \{100, 150, \cdots, 300\}$.
2. Bandwidth for `kernel`; choice of parametric family for `flexsurv` and `blackboost`; number of trees in `blackboost`.

take the form of Beta PDFs, and λ_3 is the hazard of the log-normal distribution. As will be explained, the first two cases naturally favour `kernel` while the third one favours `flexsurv`.

To investigate the robustness of `BoXHED` to noise in a high dimensional setting, we also add up to 40 irrelevant covariates to each hazard function. The trajectories of the covariates are simulated as piecewise-constant paths with values drawn from $U(0, 1]$.

For the training set we draw 5,000 sample trajectories, and we also draw 5,000 for the test set.

Results. Table 1 presents the L^2-errors for the hazard estimators when applied to the simulated datasets. The main takeaways are:

- `BoXHED` always outperforms `kernel` when irrelevant covariates are present. The methods are comparable when no irrelevant covariates are present, with `kernel` having the edge in the first two cases λ_1 and λ_2. This is because the kernel function $K(u)$ is a location- and scale-transformed Beta PDF, which is also the functional form for λ_1 and λ_2. It is therefore unsurprising that `kernel` is able to approximate λ_1 and λ_2 better than regression trees. The minuscule edge that `kernel` enjoys for λ_4 is likely due to the fact that it was tuned directly to the test data.

- For λ_3, `flexsurv` performs the best, followed closely by `BoXHED`. The reason for `flexsurv`'s outperformance is due to the fact that it includes the log-normal distribution as one of its parametric options, so it is correctly specified for λ_3.

- Neither `BoXHED` nor `blackboost` are affected much by irrelevant covariates, while `kernel`'s performance drops dramatically when irrelevant covariates are added. These findings are in line with the well known fact that kernel smoothing suffers from the curse of dimensionality, while boosted trees automatically perform variable selection.

5. Discussion

Survival data with potentially high-dimensional time-dependent covariates are becoming increasingly common in applications, yet there is a paucity of theoretically sound methods for analyzing them in a nonparametric way. Our proposed hazard estimator adds to this literature in a meaningful way, and we have made the `BoXHED` implementation available on GitHub to extend the machine learning toolbox for survival analysis. The simulation results here illustrate the value of `BoXHED` for time-dependent covariate survival problems.

References

O. O. Aalen. Nonparametric inference for a family of counting processes. *Annals of Statistics*, 6(4):701–726, 1978.

A. Bellot and M. van der Schaar. Boosted trees for risk prognosis. In *Machine Learning for Healthcare Conference*, pages 2–16, 2018.

A. Bellot and M. van der Schaar. Boosting transfer learning with survival data from heterogeneous domains. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 57–65, 2019.

H. Binder and M. Schumacher. Allowing for mandatory covariates in boosting estimation of sparse high-dimensional survival models. *BMC Bioinformatics*, 9(1):14, 2008.

P. Blanche, M. W. Kattan, and T. A. Gerds. The c-index is not proper for the evaluation of-year predicted risks. *Biostatistics*, 20(2):347–357, 2019.

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

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

T. Hothorn. Transformation boosting machines. *Statistics and Computing*, pages 1–12, 2019.

H. Ishwaran, U. B. Kogalur, E. H. Blackstone, and M. S. Lauer. Random survival forests. *The Annals of Applied Statistics*, 2(3):841–860, 2008.

D. Jarrett, J. Yoon, and M. van der Schaar. MATCH-Net: Dynamic prediction in survival analysis using convolutional neural networks. *arXiv preprint arXiv:1811.10746*, 2018.

Changhee Lee, William Zame, Ahmed Alaa, and Mihaela Schaar. Temporal quilting for survival analysis. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 596–605, 2019.

D. K. K. Lee, N. Chen, and H. Ishwaran. Boosted nonparametric hazards with time-dependent covariates. *Annals of Statistics (forthcoming)*, 2021.

H. Li and Y. Luan. Boosting proportional hazards models using smoothing splines, with applications to high-dimensional microarray data. *Bioinformatics*, 21(10):2403–2409, 2005.

J. P. Nielsen and O. B. Linton. Kernel estimation in a nonparametric marker dependent hazard model. *Annals of Statistics*, 23(5):1735–1748, 1995.

M. L. G. Pérez, L. Janys, M. D. Martinez-Miranda, and J. P. Nielsen. Bandwidth selection in marker dependent kernel hazard estimation. *Computational Statistics and Data Analysis*, 68:155–169, 2013.

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

K. Ren, J. Qin, L. Zheng, Z. Yang, W. Zhang, L. Qiu, and Y. Yu. Deep recurrent survival analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4798–4805, 2019.

G. Ridgeway. The state of boosting. *Computing Science and Statistics*, 31:172–181, 1999.

X. Wang, A. Pakbin, B. J. Mortazavi, H. Zhao, and D. K. K. Lee. BoXHED: Boosted exact hazard estimator with dynamic covariates. In *International Conference on Machine Learning*, pages 9973–9982. PMLR, 2020.