


Survival Regression with Accelerated Failure Time Model in XGBoost

Avinash Barnwal, Hyunsu Cho & Toby Hocking


To cite this article: Avinash Barnwal, Hyunsu Cho & Toby Hocking (2022) Survival Regression with Accelerated Failure Time Model in XGBoost, Journal of Computational and Graphical Statistics, 31:4, 1292-1302, DOI: [10.1080/10618600.2022.2067548](https://doi.org/10.1080/10618600.2022.2067548)

To link to this article: <https://doi.org/10.1080/10618600.2022.2067548>

 View supplementary material [↗](#)

 Published online: 24 May 2022.

 Submit your article to this journal [↗](#)

 Article views: 1290

 View related articles [↗](#)

 View Crossmark data [↗](#)

 Citing articles: 23 View citing articles [↗](#)



Survival Regression with Accelerated Failure Time Model in XGBoost

Avinash Barnwal^a, Hyunsu Cho^b, and Toby Hocking^c

^aDepartment of Statistics, Stony Brook University, Stony Brook, NY; ^bNVIDIA, Santa Clara, CA; ^cSchool of Informatics, Computing, and Cyber Systems, Northern Arizona University, Flagstaff, AZ

ABSTRACT

Survival regression is used to estimate the relation between time-to-event and feature variables, and is important in application domains such as medicine, marketing, risk management, and sales management. Nonlinear tree based machine learning algorithms as implemented in libraries such as XGBoost, scikit-learn, LightGBM, and CatBoost are often more accurate in practice than linear models. However, existing state-of-the-art implementations of tree-based models have offered limited support for survival regression. In this work, we implement loss functions for learning accelerated failure time (AFT) models in XGBoost, to increase the support for survival modeling for different kinds of label censoring. We demonstrate with real and simulated experiments the effectiveness of AFT in XGBoost with respect to a number of baselines, in two respects: generalization performance and training speed. Furthermore, we take advantage of the support for NVIDIA GPUs in XGBoost to achieve substantial speedup over multi-core CPUs. To our knowledge, our work is the first implementation of AFT that uses the processing power of NVIDIA GPUs. Starting from the 1.2.0 release, the XGBoost package natively supports the AFT model. The addition of AFT in XGBoost has had significant impact in the open source community, and a few statistics packages now use the XGBoost AFT model. Supplementary materials for this article are available online.

ARTICLE HISTORY

Received November 2020
Accepted April 2022

KEYWORDS

Gradient boosting; GPU computing; Open source; Survival analysis; XGBoost

1. Introduction

Survival analysis is a prominent subfield of statistics, where the goal is to model time duration to a given event (e.g., death). Given the nature of time-to-event data, labels may not be completely observed and only censored labels are given. For data points whose label is censored, the exact label y is not known but only a range (y, \bar{y}) that contains it. The topic has drawn a large body of research literature in the last few decades. See Wang, Li, and Reddy (2019) for a general survey.

The Cox proportional hazards (Cox-PH) model (Cox 1972) is one of the most commonly used models in survival analysis. The model estimates the hazard function $h(t)$, which is defined to be the likelihood of the event occurring at time t given that no event has occurred before time t . The Cox-PH model is of form $h(t, \mathbf{x}) = h_0(t) \exp(\langle \mathbf{w}, \mathbf{x} \rangle)$, where the baseline hazard function $h_0(t)$ depends only on time and the features \mathbf{x} have multiplicative effects on h . Given the parameters \mathbf{w} , it is clear which of the normalized features \mathbf{x} has the largest effect on survival. However, it is nontrivial to predict time-to-event $\hat{y}(\mathbf{x})$ in the Cox-PH model. We would need to estimate the baseline hazard function $h_0(t)$ using a nonparametric estimator known as Breslow's estimator (Breslow 1972; Allison 2010). The computation of Breslow's estimator requires access to the full training data and is computationally expensive for big data.

The accelerated failure time (AFT) model is another well-known method for survival analysis, although perhaps less often

used than Cox-PH. We choose to explore AFT in this article for two primary reasons. First, we would like to not only analyze model parameters (coefficients) but also perform predictive analysis. While Cox-PH gives relative importance of features, it does not yield a usable prediction \hat{y} easily (Allison 2010). With the AFT model, we can predict unknown labels using only the fitted parameters and a feature vector. Second, the AFT model may provide a better fit when proportional hazard assumption does not hold (Faruk 2018).

Miller (1976) proposed the AFT model for the first time, and later Buckley and James (1979) refined it to obtain an asymptotically consistent estimator using the least squares approach. Khan and Shaw (2016) and Mimi and Khan (2021) discuss methods for enabling variable selection to fit AFT models from high-dimensional data. See Wei (1992) and Chiou, Kang, and Yan (2014) for overviews on the topic of AFT models.

Tree-based models have shown better performance than linear models in terms of detecting complex and nonlinear patterns in the feature variables. The gradient boosting algorithm (Friedman 2001) fits an additive ensemble of decision trees in a stepwise fashion to greedily optimize a general class of loss functions $\ell(y, \hat{y})$. Gradient boosting is widely used due to its simplicity and predictive performance. The algorithm produces an ensemble of decision trees and exhibits many desirable properties as a statistical model, such as being slow to overfitting and having asymptotic convergence guarantees (Bühlmann

and Hothorn 2007; Zhang and Yu 2003). Gradient boosting is versatile, as it can optimize a general class of loss function $\ell(y, \hat{y})$ where y represents the true label and \hat{y} the predicted label. It has been successfully used in classification (Friedman, Hastie, and Tibshirani 2000), document ranking (Burgess 2010), structured prediction (Chen et al. 2015) and other applications. Today, there are several scalable, efficient software packages that implement gradient boosting, including XGBoost (Chen and Guestrin 2016), LightGBM (Ke et al. 2017), Scikit-Learn (Pedregosa et al. 2011), and Catboost (Prokhorenkova et al. 2018).

XGBoost is a fast implementation of gradient boosting that speeds up convergence by using the second-order partial derivative of the loss function. XGBoost is able to integrate with a variety of programming environments such as R and Python and integrates with frameworks for distributed computing, such as Dask and Apache Spark. Integration of AFT with XGBoost therefore, makes survival analysis easier in the big data setting.

There are a few previous implementations of survival analysis in tree based models. Schmid and Hothorn (2008) uses boosting framework for AFT and considers the negative log-likelihood as loss function. There are also other tree based survival models such as Random Survival Trees (Ishwaran et al. 2008), Cox-Boosting (Binder and Schumacher 2008), Bagging Survival Trees (Hothorn et al. 2004), Scikit-Survival (Pölsterl 2020), and Cox-PH in XGBoost (Lundberg, Erion, and Lee 2019). Most of the models are limited to right-censored outcomes. Maximum Margin Interval Trees (Drouin, Hocking, and Laviolette 2017) support interval-censored labels.

Survival analysis is broadly useful in a variety of applications, such as survival prediction of cancer patients (Viganò et al. 2000), customer churn (Van den Poel and Larivière 2004), credit scoring (Dirick, Claeskens, and Baesens 2017), failure times of mechanical systems (Susto et al. 2015; Barabadi, Barabady, and Markeset 2010). However, binary machine learning classifiers have been often used where survival methodology is applicable, due to concerns about predictive accuracy (Kvamme, Borgan, and Scheel 2019). For example, Vaid et al. (2020) uses XGBoost binary classifiers to predict whether COVID-19 patients will develop complications in a given time frame, achieving substantially better AUC-ROC and AUC-PR than generalized linear models. While binary classifiers may provide for a state-of-art predictive accuracy, one loses flexibility that comes from directly modeling time duration to events: one is forced to decide pre-determined duration(s) where an event is to occur or not. AFT in XGBoost addresses these challenges in the following two ways. First, the model is able to capture nonlinear patterns in the data. Second, the model readily produces survival time estimates; to compute predictions, we only need the fitted model parameters and a feature vector.

Summary of novel contributions. We propose a novel adaptation of the AFT model to integrate with XGBoost. Our implementation supports all modes of label censoring, including interval-censoring. We run experiments with real and simulated datasets to demonstrate the generalization performance of the AFT model in XGBoost. Furthermore, we are able to accelerate training by using XGBoost’s built-in support for NVIDIA GPUs.

2. AFT in XGBoost

The original AFT model takes the following form:

$$\ln Y = \langle \mathbf{w}, \mathbf{x} \rangle + \sigma Z,$$

where \mathbf{x} represents the input features, \mathbf{w} the coefficients, Y the survival time (the output label), and Z a random variable of a known probability distribution. Both Y and Z are random variables. Note that this model is a generalized form of a linear regression model $Y = \langle \mathbf{w}, \mathbf{x} \rangle$. In order to make AFT work with gradient boosting, we revise the model as follows:

$$\ln Y = \mathcal{T}(\mathbf{x}) + \sigma Z, \tag{1}$$

where $\mathcal{T}(\mathbf{x})$ represents the output from the decision tree ensemble, given input \mathbf{x} .

2.1. Derivation of AFT Loss Function

XGBoost optimizes a twice-differentiable convex loss function $\ell(\cdot, \cdot)$ in its second-order method of gradient boosting (Chen and Guestrin 2016). We will now define a suitable loss function ℓ_{AFT} to represent the AFT model. Let $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ denote the training data, and let Y_1, \dots, Y_n denote random variables iid with the distribution for Y . Let f_Y and F_Y denote the probability density function (PDF) and the cumulative distribution function (CDF) for Y , respectively. The likelihood function for \mathcal{D} is the product of probability densities f_Y for individual data points:

$$L(\mathcal{D}) = \mathbb{P}[Y_1 = y_1, \dots, Y_n = y_n] = \prod_{i=1}^n \mathbb{P}[Y_i = y_i] = \prod_{i=1}^n f_Y(y_i).$$

As commonly done in machine learning literature, we maximize log-likelihood instead:

$$\ln L(\mathcal{D}) = \sum_{i=1}^n \ln \mathbb{P}[Y_i = y_i] = \sum_{i=1}^n \ln f_Y(y_i). \tag{2}$$

Since we do not know y_i for some data points, due to label censoring, we revise the likelihood function (2) to take account of the censored labels:

$$\begin{aligned} \ln L(\mathcal{D}) &= \underbrace{\sum \ln \mathbb{P}[Y_i = y_i]}_{\text{uncensored label}} + \underbrace{\sum \ln \mathbb{P}[y_i \leq Y_i \leq \bar{y}_i]}_{\text{censored label with } y_i \in [y_i, \bar{y}_i]} \\ &= \underbrace{\sum \ln f_Y(y_i)}_{\text{uncensored label}} + \underbrace{\sum \ln (F_Y(\bar{y}_i) - F_Y(y_i))}_{\text{censored label with } y_i \in [y_i, \bar{y}_i]}, \end{aligned}$$

where y_i and \bar{y}_i are lower and upper bounds for the label y_i , respectively. Note that \bar{y}_i may be infinity, to indicate right-censored labels. See Table 1 for full list of censoring types. We are now ready to define the loss function ℓ_{AFT} .

Table 1. List of label censoring types.

Label censoring	Lower bound (y_i)	Upper bound (\bar{y}_i)
Right-censored	Finite nonnegative	$+\infty$
Left-censored	0	Finite nonnegative
Interval-censored	Finite nonnegative	Finite nonnegative

Table 2. Probability distributions for Z .

Distribution	PDF ($f_Z(z)$)	CDF ($F_Z(z)$)	$f'_Z(z)$	$f''_Z(z)$
Normal	$\frac{\exp(-z^2/2)}{\sqrt{2\pi}}$	$\frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{z}{\sqrt{2}} \right) \right)$	$-zf_Z(z)$	$(z^2 - 1)f_Z(z)$
Logistic	$\frac{e^z}{(1 + e^z)^2}$	$\frac{e^z}{1 + e^z}$	$f_Z(z)(1 - e^z)$	$f_Z(z)(e^{2z} - 4e^z + 1)$
Extreme ¹	$\frac{(1 + e^z)^2}{e^z e^{-\exp z}}$	$\frac{1 + e^z}{1 - e^{-\exp z}}$	$\frac{1 + e^z}{(1 - e^z)f_Z(z)}$	$\frac{(1 + e^z)^2}{(e^{2z} - 3e^z + 1)f_Z(z)}$

NOTE: ¹Also known as the Gumbel (minimum) distribution. See Therneau (2015).

Definition 1 (Loss function for AFT survival regression).

$$\ell_{\text{AFT}}(y, \mathcal{T}(\mathbf{x})) = \begin{cases} -\ln f_Y(y) & \text{if } y \text{ is not censored} \\ -\ln(F_Y(\bar{y}) - F_Y(y)) & \text{if } y \text{ is censored with } y \in [y, \bar{y}] \end{cases} \quad (3)$$

Under this definition, the sum of losses $\sum_{i=1}^n \ell(y_i, \mathcal{T}(\mathbf{x}_i))$ over the training data is identical to $-\ln L(\mathcal{D})$. Since we only know distribution of Z (not of Y), we use the following lemma:

Lemma 1 ((1.27) of Bishop (2006)). Let Y and Z be random variables. If $Y = g(Z)$ with a monotone increasing function $g(\cdot)$ that is suitably smooth, the PDF and CDF of Y are expressed in terms of the PDF and CDF of Z as follows:

$$f_Y(y) = f_Z(g^{-1}(y)) \cdot \frac{d}{dy} g^{-1}(y) \quad F_Y(y) = F_Z(g^{-1}(y))$$

We apply Lemma 1 to (3) with $g(Z) = \exp(\mathcal{T}(\mathbf{x}) + \sigma Z)$ to get the following formula for ℓ_{AFT} :

Definition 2 (Loss function for AFT survival regression, in terms of known PDF and CDF).

$$\ell_{\text{AFT}}(y, \mathcal{T}(\mathbf{x})) = \begin{cases} -\ln \left[f_Z(s(y)) \cdot \frac{1}{\sigma y} \right] & \text{if } y \text{ is not censored} \\ -\ln [F_Z(s(\bar{y})) - F_Z(s(y))] & \text{if } y \text{ is censored with } y \in [y, \bar{y}], \end{cases}$$

where f_Z and F_Z are given by Table 2 and $s(y) = (\ln y - \mathcal{T}(\mathbf{x}))/\sigma$ is a link function.

See Figure 1 for a geometric representation. Since the prediction $\mathcal{T}(\mathbf{x})$ from the tree ensemble model approximates the log survival time $\ln y$, we use the link function $s(y, \mathcal{T}(\mathbf{x})) = (\ln y - \mathcal{T}(\mathbf{x}))/\sigma$ in Definition 2 as a convenient measure for the distance between the prediction and the log survival time. Although s is a function of two variables, we will use $s(y)$ as a shorthand for $s(y, \mathcal{T}(\mathbf{x}))$ to save space.

2.2. Gradient and Hessian of the AFT Loss

The gradient boosting algorithm in XGBoost uses the gradient and Hessian of the loss function, which are first and second partial derivatives of $\ell(y, \mathcal{T}(\mathbf{x}))$ with respect to the second input

$\mathcal{T}(\mathbf{x})$. To express partial derivatives in a concise manner, define a single-letter variable $u = \mathcal{T}(\mathbf{x})$ as an alias for the output from the tree ensemble model. The gradient and Hessian of the AFT loss function are as follows¹:

Definition 3 (Gradient and Hessian of AFT loss).

$$\frac{\partial \ell_{\text{AFT}}}{\partial u} \Big|_{y,u} = \begin{cases} \frac{f'_Z(s(y))}{\sigma f_Z(s(y))} & \text{if } y \text{ is not censored} \\ \frac{f_Z(s(\bar{y})) - f_Z(s(y))}{\sigma [F_Z(s(\bar{y})) - F_Z(s(y))]} & \text{if } y \text{ is censored with } y \in [y, \bar{y}] \end{cases} \quad (4)$$

$$\frac{\partial^2 \ell_{\text{AFT}}}{\partial u^2} \Big|_{y,u} = \begin{cases} -\frac{f_Z(s(y))f''_Z(s(y)) - f'_Z(s(y))^2}{\sigma^2 f_Z(s(y))^2} & \text{if } y \text{ is not censored} \\ \frac{-[F_Z(s(\bar{y})) - F_Z(s(y))][f'_Z(s(\bar{y})) - f'_Z(s(y))] + [f_Z(s(\bar{y})) - f_Z(s(y))]^2}{\sigma^2 [F_Z(s(\bar{y})) - F_Z(s(y))]^2} & \text{if } y \text{ is censored,} \end{cases} \quad (5)$$

where f'_Z and f''_Z are the first and second derivatives of the PDF f_Z , respectively, and $s(y) = (\ln y - u)/\sigma$ is defined the same way as in Definition 2. See Table 2 to look up f'_Z and f''_Z for the three known distributions.

Proof. The first- and second-order partial derivatives of ℓ_{AFT} may be derived using basic rules of Calculus. Consult Section 1 of the supplementary materials for the full proof. \square

2.3. Regularization for the AFT Loss

The Equations (2), (4), and (5) may suffer from numerical instabilities when the prediction $u = \mathcal{T}(\mathbf{x})$ from the tree ensemble model is far away from the true log survival time $\ln y \in [\ln y, \ln \bar{y}]$. There are three causes for the numerical instabilities:

¹For left- and right-censored labels, let $f_Z(-\infty) = f_Z(\infty) = 0$ and $F_Z(-\infty) = 0, F_Z(+\infty) = 1$.

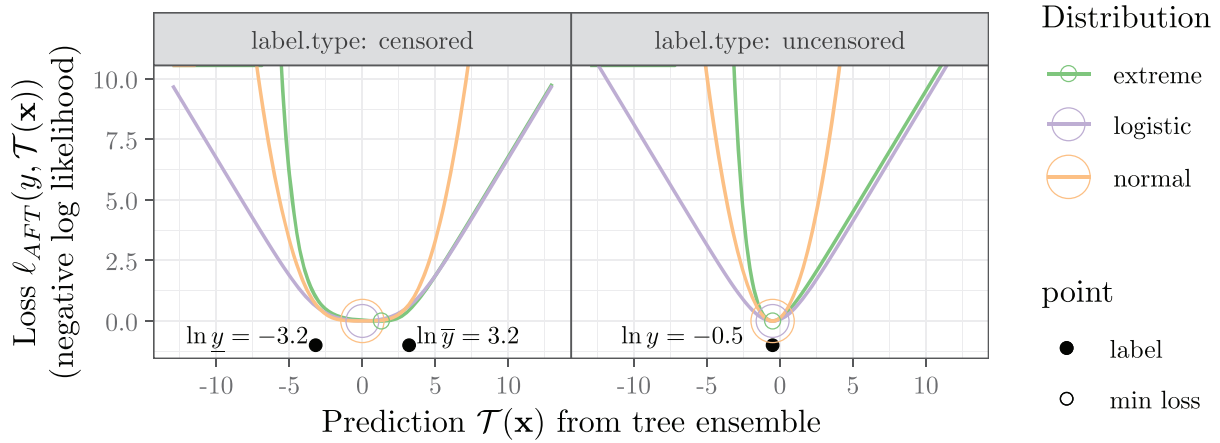


Figure 1. Geometric interpretation of the accelerated failure time (AFT) loss (colored curves), using three distributions (normal, logistic, extreme) with scale parameter $\sigma = 1$. Log survival times are shown using solid black dots, and loss function minima are shown using open colored circles. Note that the prediction $\mathcal{T}(\mathbf{x})$ from the tree ensemble model is in the same scale as the log survival time $\ln y$. *Left:* for censored data the loss function is defined as the negative log of the difference of cumulative distribution function values. The example shown has finite upper and lower limits, for which the minimum of the logistic/normal loss occurs at the midpoint between the two limits, whereas it occurs at a greater value for the extreme distribution. *Right:* for uncensored data the loss function is defined as the negative log of the density function, so the normal loss is the usual square loss with symmetric quadratic tails. The logistic loss has symmetric linear tails, whereas the asymmetric extreme loss has a linear upper tail and an exponential lower tail.

- Zero passed to the logarithm function $\ln(\cdot)$: the difference term $F_Z(s(\bar{y})) - F_Z(s(\underline{y}))$ in (2) becomes nearly zero when the prediction $u = \mathcal{T}(\bar{\mathbf{x}})$ is far away from the true log survival time $\ln y \in [\ln \underline{y}, \ln \bar{y}]$. Passing zero to the logarithm results in a NAN (Not-a-Number).
- Zero denominator: the denominators in (4) and (5) for censored data contain the difference term $F_Z(s(\bar{y})) - F_Z(s(\underline{y}))$, which can be nearly zero for the same reason as above. Zero denominator results in a NAN (Not-a-Number).
- Large number passed to the exponential function $\exp(\cdot)$: the PDF and CDF of the logistic and extreme distributions contain the exponential function. Since 64-bit floating-point variables in a C++ program hold up to 10^{308} , the exponential function yields in an INF (infinity) even for moderately large inputs.

Refer to the IEEE 754 standard (IEEE 2019) to learn more about the special representations of floating-point values, such as INFs and NANs. In order to eliminate INFs and NANs, we apply regularization in two places.

Regularization for the loss function (2). We replace the difference term $F_Z(s(\bar{y})) - F_Z(s(\underline{y}))$ with $\epsilon = 10^{-12}$ whenever the difference term is smaller than ϵ .

Regularization for the gradient (4) and the Hessian (5). We explicitly define values of the gradient and Hessian at the limit $u \rightarrow \pm\infty$. Whenever a numerical instability is detected due to u being far away from the true log survival time, we set the gradient and Hessian values according to Table 3. We clip all gradients to ± 15 , as large gradients cause numerical difficulties in XGBoost. In addition, we ensure that the Hessian value is at least 10^{-16} for all data points, because data points with zero Hessian are ignored by XGBoost. Regularization forces XGBoost to consider every data point to a certain extent.

Table 3. Specification of the gradient and Hessian values ($\partial \ell / \partial u, \partial^2 \ell / \partial u^2$) as $u \rightarrow \pm\infty$.

Dist. for Z	Label type	Uncensored		Right-censored	
		$u \rightarrow -\infty$	$u \rightarrow +\infty$	$u \rightarrow -\infty$	$u \rightarrow +\infty$
Normal	$\partial \ell / \partial u$	-15	15	-15	0
	$\partial^2 \ell / \partial u^2$	$1/\sigma^2$	$1/\sigma^2$	$1/\sigma^2$	10^{-16}
Logistic	$\partial \ell / \partial u$	$-1/\sigma$	$1/\sigma$	$-1/\sigma$	0
	$\partial^2 \ell / \partial u^2$	10^{-16}	10^{-16}	10^{-16}	10^{-16}
Extreme	$\partial \ell / \partial u$	-15	$1/\sigma$	-15	0
	$\partial^2 \ell / \partial u^2$	15	10^{-16}	15	10^{-16}
Dist. for Z	Label type	Left-censored		Interval-censored	
		$u \rightarrow -\infty$	$u \rightarrow +\infty$	$u \rightarrow -\infty$	$u \rightarrow +\infty$
Normal	$\partial \ell / \partial u$	0	15	-15	15
	$\partial^2 \ell / \partial u^2$	10^{-16}	$1/\sigma^2$	$1/\sigma^2$	$1/\sigma^2$
Logistic	$\partial \ell / \partial u$	0	$1/\sigma$	$-1/\sigma$	$1/\sigma$
	$\partial^2 \ell / \partial u^2$	10^{-16}	10^{-16}	10^{-16}	10^{-16}
Extreme	$\partial \ell / \partial u$	0	$1/\sigma$	-15	$1/\sigma$
	$\partial^2 \ell / \partial u^2$	10^{-16}	10^{-16}	15	10^{-16}

3. Experiments

3.1. Effectiveness of AFT for Interval-Censored Data

We measure the effectiveness of the XGBoost AFT model for interval-censored data. We define the accuracy metric for data with interval-censored labels as follows:

$$\text{Accuracy}(\mathcal{D}) = \frac{|\{i : \mathcal{T}(\mathbf{x}_i) \in [\ln \underline{y}_i, \ln \bar{y}_i]\}|}{|\mathcal{D}|},$$

that is, the fraction of data points for which the prediction from the tree ensemble model falls between the lower and upper bounds for the true log survival time.

The XGBoost AFT model is compared to three baselines:

survreg Unregularized linear model with AFT loss functions (Therneau 2015) on principal components, with the number of components selected using cross-validation.

penaltyLearning L1-regularized linear model with squared hinge loss (Rigaill et al. 2013), with the degree of L1 regularization selected by cross-validation.

MMIT Max Margin Interval Trees (Drouin, Hocking, and Laviolette 2017), which generalizes the well-known Classification and Regression Tree (CART) algorithm of Breiman et al. (1984) to censored outputs. The tree depth is selected using cross-validation.

We perform nested cross-validation to estimate the generalization performance of the model as well as the hyperparameter search procedure. We use 5-fold internal cross-validation to evaluate multiple hyperparameter combinations. Each combination is judged using the mean validation accuracy over the 5-fold. (The mean validation accuracy is also used to determine the number of boosting rounds.) We then perform 4-fold external cross-validation to quantify the generalization performance of the training procedure, as follows: we fit a new model using the best hyperparameter combinations, using all data points except the held-out test set. The test accuracy is evaluated with the held-out test set. For hyperparameter search, we run 100 trials in the random search; see Section 3.3 for details.

Experiments in Sections 3.1.1 and 3.1.2 were conducted using a workstation with one Intel Core i7-7800X CPU (3.50 GHz, 6 cores) and two DDR4 RAMs (16 GB each, 2133 MHz), running Ubuntu 18.04 LTS.

3.1.1. Interval-Censored Data from Supervised Change-point Detection Problems

To test our algorithm in real datasets with interval-censored outputs, we consider a benchmark of supervised peak detection problems in genomic ChIP-seq data (Rigaill et al. 2013; Hocking et al. 2016). Each of the 10 datasets in Table 4 corresponds to a set of high-throughput DNA sequencing experiments. Expert biologists manually labeled each dataset to indicate locations with and without peaks; these labels are used to compute the peak detection error rate. The goal of the ChIP-seq peak detection is to automatically locate peaks given a new DNA sequence, such that peak detection error rate is minimal. Each dataset is named like H3K4me3_PGP_immune:

- The first field (H3K4me3) identifies the assay type. Consult the McGill Epigenomics Mapping Centre website² for the full list of assay types and their meaning.
- The second field (PGP) is the initials of the expert biologist who provided the labels.
- The last field (immune) identifies a sample set. Consult Hocking et al. (2016) for more information.

A univariate signal is computed for each sample by computing coverage frequency of aligned DNA sequence reads at each positions of the reference genome sequence. To detect peaks in the univariate signal, we use an changepoint detection algorithm with learned penalty functions (Rigaill et al. 2013). Briefly, the penalty parameter λ of the changepoint detection algorithm controls the number of distinct segments/peaks detected. It is possible to compute a range of optimal penalty values $[\lambda_*, \bar{\lambda}_*]$ that are optimal in terms of the expert-provided labels; that

Table 4. Dimensions of ChIP-seq datasets and descriptive statistics.

Dataset	Rows	Columns	min.log .lambda	max.log .lambda
(1) ATAC_JV_adipose	465	36	8.581	10.470
(2) CTCF_TDH_ENCODE	182	36	10.246	12.643
(3) H3K27ac-H3K4me3_TDHAM_BP	2008	36	7.674	9.641
(4) H3K27ac_TDH_some	95	36	9.318	10.365
(5) H3K36me3_AM_immune	420	36	8.955	12.723
(6) H3K27me3_RL_cancer	171	36	14.332	16.192
(7) H3K27me3_TDH_some	43	36	10.617	11.389
(8) H3K36me3_TDH_ENCODE	78	36	8.147	9.634
(9) H3K36me3_TDH_immune	84	36	10.939	13.003
(10) H3K36me3_TDH_other	40	36	9.742	12.389

NOTE: The log lambda values given above are averages over the dataset.

is, setting λ to any value in $[\lambda_*, \bar{\lambda}_*]$ will minimize the label error rate. We cast the peak detection problem into an interval-censored regression problem as follows. The univariate signal is used to compute a feature vector $\mathbf{x} \in \mathbb{R}^{36}$ that stores various summary statistics, such as percentiles, of the signal. The range of optimal penalty values $[\lambda_*, \bar{\lambda}_*]$ is taken to be an interval-censored label.

We preprocess the data as follows: we apply the exponential function $\exp(\cdot)$ to the output labels `min.log.lambda` and `max.log.lambda` to obtain the nonnegative interval-censored labels `min.lambda` and `max.lambda`. Then we remove all feature columns that either (i) had at least one missing value or (ii) had zero variance.

Figure 2(a) shows the generalization performance (test accuracy) and run time of XGBoost and the baseline packages. XGBoost exhibits competitive generalization performance on par with SurvReg, penaltyLearning, and MMIT. In addition, XGBoost is fast: its run time approaches that of SurvReg and penaltyLearning and significantly smaller than that of MMIT. Considering that SurvReg and penaltyLearning are linear models and MMIT nonlinear, the run-time performance speaks to the efficiency of XGBoost.

3.1.2. Synthetic Interval-Censored Data Generated from Known Distributions

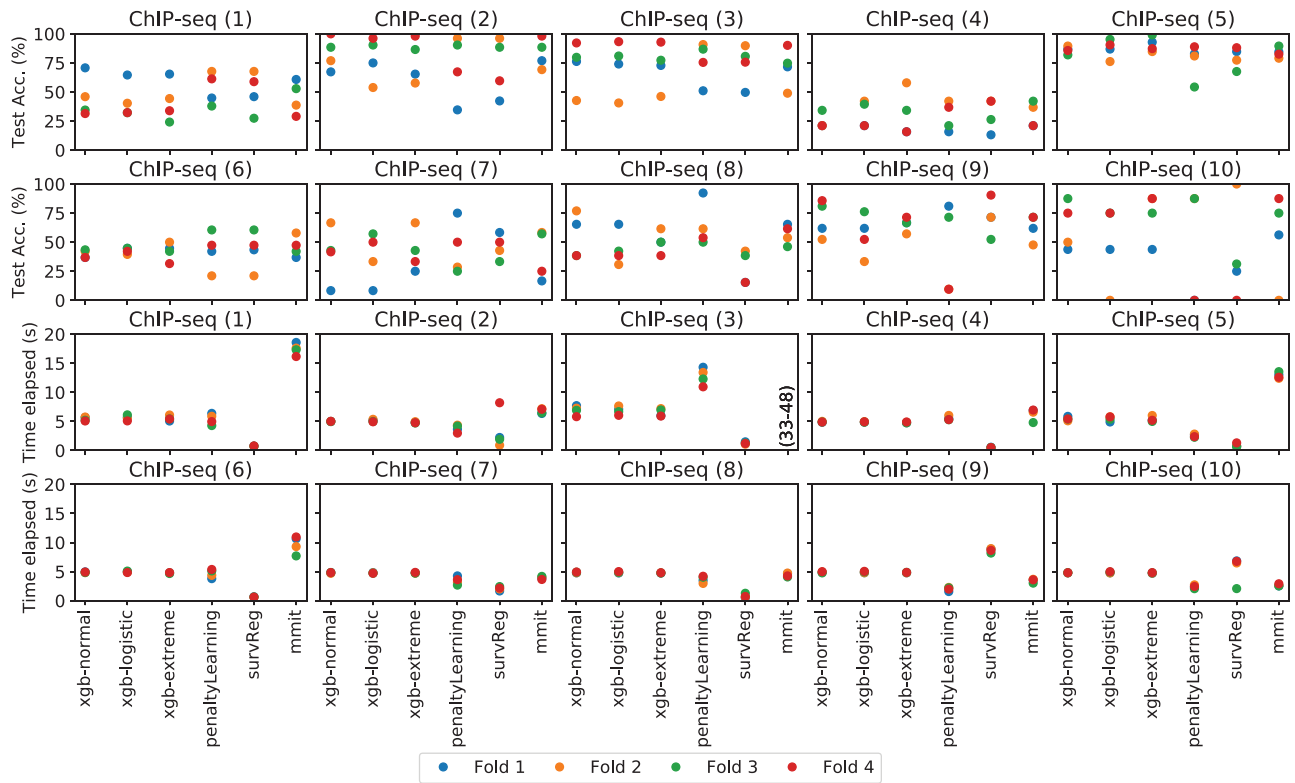
Drouin, Hocking, and Laviolette (2017) generated synthetic interval-censored data based on three kind of features: sine, absolute and linear. It has a mix of nonlinear and linear features having 200 samples and 20 features in each dataset. We extend it with three more datasets having more complex nonlinear features. We use a random number generator to generate interval-censored data as follows.

First, generate the feature vectors $\mathbf{x} \in \mathbb{R}^{20}$ by sampling from the uniform distribution $U([0, 10])$. Second, draw 10 values randomly from the normal distribution $\mathcal{N}(f(\mathbf{x}), 0.3)$ where the mean is determined with a function $f: \mathbb{R}^{20} \rightarrow \mathbb{R}$ that maps the feature vector \mathbf{x} to a real value. Third, out of the 10 values, choose the smallest as the lower bound \underline{y} and the largest as the upper bound \bar{y} . Lastly, add a small noise to both the interval bounds by sampling a value from $\mathcal{N}(0, 0.2)$ and adding it to \underline{y} and \bar{y} .

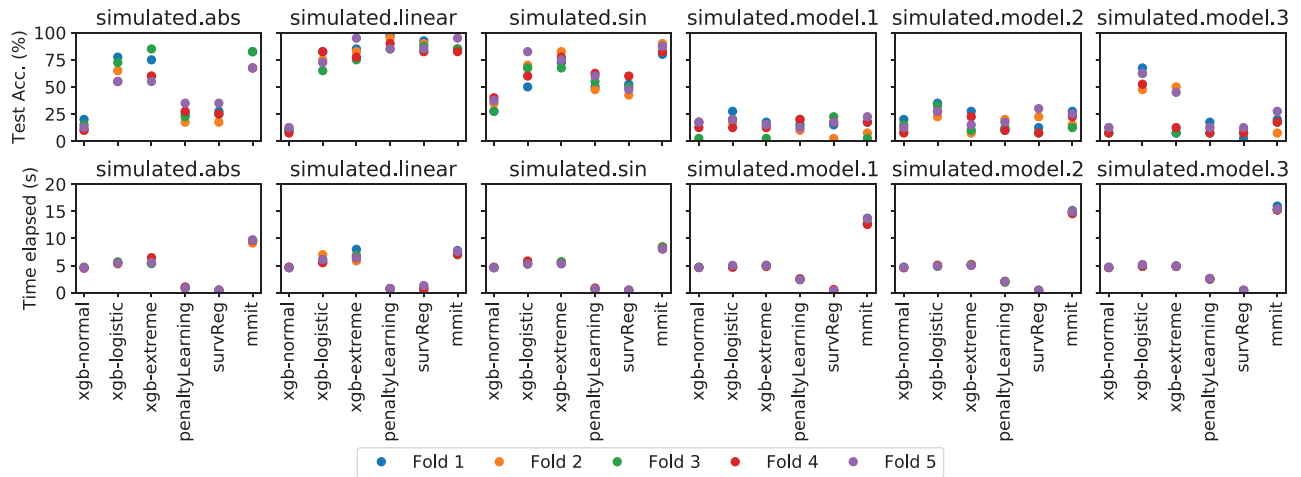
Each generated dataset was named after the choice for f :

- `simulated.sin`: $f(\mathbf{x}) = \sin(x_1)$
- `simulated.abs`: $f(\mathbf{x}) = |x_1 - 5|$
- `simulated.linear`: $f(\mathbf{x}) = x_1/5$

²<https://epigenomesportal.ca/edcc/index.html>.



(a) CHIP-seq data from Table 4.



(b) Simulated data.

Figure 2. Experimental results from Section 3.1: Test accuracy and run time.

- `simulated.model.1`: $f(\mathbf{x}) = x_1x_2 + x_3^2 - x_4x_7 + x_8x_{10} - x_6^2$
- `simulated.model.2`: $f(\mathbf{x}) = -\sin(2x_1) + x_2^2 + x_3 - \exp(-x_4)$
- `simulated.model.3`: $f(\mathbf{x}) = x_1 + 3x_3^2 - 2\exp(-x_5)$

We compare the performance of penalty Learning, survReg, MMIT, and XGBoost on test data of size 100. As in Section 3.1.1, we perform nested cross-validation to estimate the generalization performance of the model as well as the hyperparameter search procedure; this time, we use 5-fold for both the outer and

inner cross-validation. We reproduce the behavior in Drouin, Hocking, and Laviolette (2017), where nonlinear models like mmit better capture nonlinear patterns in simulated data than linear models do. In Figure 2(b), both XGBoost and mmit exhibit superior generalization performance (test accuracy) compared to the linear models, SurvReg and penaltyLearning. The additional run-time incurred by the nonlinear models is compensated by higher test accuracy. The difference between XGBoost and mmit is more pronounced when we look at the three simulated data we added apart from those from Drouin, Hocking, and Laviolette (2017). XGBoost runs faster than mmit,

up to 3x, and shows higher test accuracy. In particular, for `simulated.model.3`, XGBoost achieves 58.5% mean test accuracy, whereas `mmit` achieves 18%.

3.2. Effectiveness of AFT on Right-Censored Data

We measure the effectiveness of the XGBoost AFT model for right-censored data using Uno's C -statistics (Uno et al. 2011), which is a modified form of Harell's Concorance Index (Harrell Jr. et al. 1982). Uno's C -statistics is an unbiased nonparametric estimator for the following ranking metric:

$$C = \mathbb{P}[\mathcal{T}(\mathbf{x}_i) < \mathcal{T}(\mathbf{x}_j) | y_i < y_j, y_i < \tau].$$

The τ constant is chosen judiciously in order to truncate outlier labels when estimating C . In this experiment, we set τ to the 80th percentile of the observed survival time. We use the implementation of Uno's C -statistics from the Scikit-Survival package (Pölsterl 2020).

The XGBoost AFT model is compared to two baselines:

XGBoost-Cox Cox-PH model from the XGBoost package (Lundberg, Erion, and Lee 2019), enabled by setting configuration `objective='survival:cox'`.

Scikit-Survival Cox-PH linear model from the Scikit-Survival package (Pölsterl 2020)

As in Section 3.1, we use nested cross-validation to assess the generalization performance of the model as well as the hyperparameter search procedure. For hyperparameter search, we run 100 trials in the random search; see Section 3.3 for details.

3.2.1. Synthetic Data with a Mix of Uncensored and Right-Censored Labels

Using a random number generator, we generate synthetic data consisting a mix of uncensored and right-censored labels, as follows³:

1. Draw a feature vector $\mathbf{x} \in \mathbb{R}^{20}$ from the uniform distribution $U([0, 1])$.
2. Draw the "risk score" $r \in \mathbb{R}$ from the normal distribution $\mathcal{N}(f(\mathbf{x}), 0.3)$ where $f(\mathbf{x}) = x_1 + 3x_3^2 - 2 \exp(-x_5)$ is a nonlinear map.
3. Draw u from the uniform distribution $U([0, 1])$.
4. Compute the ground-truth survival time $y = -\ln u / (h_0 h^r)$, where $h_0 = 0.1$ is the baseline hazard and $h = 2.0$ is the hazard ratio.
5. Simulate the effect of random censoring by drawing cutoff value c from the uniform distribution $U([0, C])$, where C is suitably chosen to induce censoring for a set fraction of data points. If $y \geq c$, the label is right-censored and we set the label range $[y, \bar{y}] = [c, +\infty)$. If $y < c$, the label is not censored and we set $[y, \bar{y}] = [y, y]$.

Repeat the steps to generate 1000 data points. The experiment result with this method of data generation is shown with label `data_gen=coxph` in Figure 3. When 20% the labels are (right-)censored, the Cox-PH model from Scikit-Survival produces slightly higher C -statistics metric than XGBoost models.

³This method is adapted from a tutorial on the Scikit-Survival package's website (Pölsterl 2020).

On the other hand, with greater amount of censoring (50%, 80%), the test C -statistics for XGBoost-AFT and XGBoost-CoxPH are similar to the test C -statistics for Scikit-Survival's Cox-PH.

We now generate data with a mix of uncensored and right-censored labels using a different method.

1. Draw a feature vector $\mathbf{x} \in \mathbb{R}^{20}$ from the uniform distribution $U([0, 1])$.
2. Draw the "risk score" $r \in \mathbb{R}$ from the normal distribution $\mathcal{N}(f(\mathbf{x}), 0.3)$ where $f(\mathbf{x}) = x_1 + 3x_3^2 - 2 \exp(-x_5)$ is a nonlinear map.
3. Compute the ground-truth survival time $y = \exp(-r)$.
4. Simulate the effect of random censoring by drawing cutoff value c from the uniform distribution $U([0, C])$. This step is analogous to Step 5 of the previous recipe.

Repeat the steps to generate 1000 data points. The experiment result with this method of data generation is shown with label `data_gen=aft` in Figure 3. When 20% of the labels are censored, XGBoost-AFT with the normal distribution produces slightly higher C -statistics metric than all other models. On the other hand, when 50% of the labels were censored, there is no clear winner; XGBoost-AFT and XGBoost-CoxPH produce similar test C -statistics as Scikit-Survival's Cox-PH. With 80% censoring, Scikit-Survival's Cox-PH produces the highest test C -statistics, and XGBoost-AFT and XGBoost-Cox are slightly behind. In all settings, XGBoost-AFT runs 2–3× faster than XGBoost-Cox-PH or Scikit-Survival's Cox-PH.

3.3. Effect of Hyperparameters on Model Performance

To measure how sensitive the generalization performance is to the choice of hyperparameters, we try an array of approaches for selecting hyperparameters. There are six relevant hyperparameters: `learning_rate`, `max_depth`, `min_child_weight`, `reg_alpha`, `reg_lambda`, and `aft_loss_distribution_scale`.⁴ The following methods are considered:

Grid search We select one or two hyperparameters out of the six and exhaustively enumerate all combinations using the grid in Table 5. If a hyperparameter is not chosen for the grid search, we assign a default value as follows: `learning_rate = 0.1`, `max_depth = 6`, `min_child_weight = 1.0`, `reg_alpha = 0.001`, `reg_lambda = 1.0`, `aft_loss_distribution_scale = 1.0`.

Random search We use Optuna (Akiba et al. 2019) to randomly sample hyperparameter combinations from the search space (Table 5). All six hyperparameters are sampled. Each search is run for 100 or 1000 combinations.

Baseline (do nothing) Choose default values for all hyperparameters and perform no search.

For the grid search, we try all possible ways of choosing one or two hyperparameters out of six. The generalization performance is measured in the aggregate with test accuracy.

⁴ σ in (1).

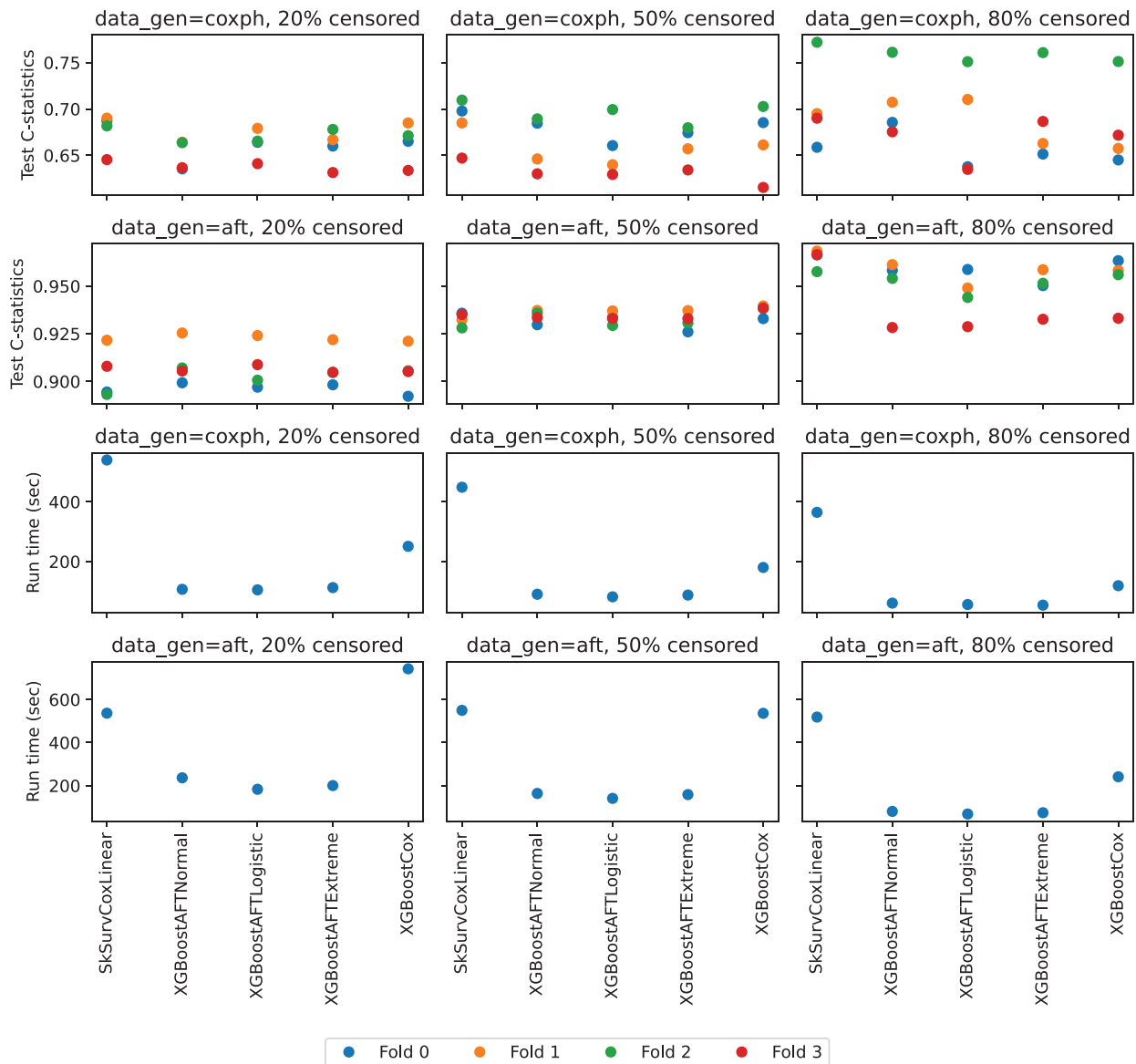


Figure 3. Experimental results from Section 3.2: Test accuracy and run time.

Table 5. Search space for hyperparameters.

Hyperparameter	Search grid
learning_rate	0.001, 0.01, 0.1, 1.0
max_depth	2, 3, 4, 5, 6, 7, 8, 9, 10
min_child_weight	0.001, 0.1, 1.0, 10.0, 100.0
reg_alpha	0.001, 0.01, 0.1, 1.0, 10.0, 100.0
reg_lambda	0.001, 0.01, 0.1, 1.0, 10.0, 100.0
aft_loss_distribution_scale	0.5, 0.8, 1.1, 1.4, 1.7, 2.0
Hyperparameter	Distribution for random search
learning_rate	log uniform in [0.001, 1.0]
max_depth	integers in [2, 10]
min_child_weight	log uniform in [0.001, 100.0]
reg_alpha	log uniform in [0.001, 100.0]
reg_lambda	log uniform in [0.001, 100.0]
aft_loss_distribution_scale	uniform in [0.5, 2.0]

As in Section 3.1.1, we perform nested cross-validation to estimate the generalization performance of the model as well as the hyperparameter search procedure. We use 4- and 5-fold for the outer and inner cross-validation, respectively. We used datasets from Sections 3.1.1 and 3.1.2.

In order to perform lots of hyperparameter search in a short amount of time, Amazon Web Services (AWS) is used to launch parallel jobs, in order to evaluate many hyperparameter search strategies. The manager EC2 instance launches hundreds of worker EC2 instances and then submits commands to execute via SSH. To ensure that all software dependencies are available to the experiment code as well as our XGBoost code, we package our code in a Docker container and host the container on Elastic Container Registry (ECR). The workers then pull the latest container image from ECR. The experiment is logged to an S3 bucket.

In all runs, the random search with 1000 trials gives the highest validation accuracy. However, high validation accuracy does not lead to high test accuracy. The grid search with one or two hyperparameters, where the number of trials is fewer than 100, yields higher test accuracy than the random search with 1000 trials. Refer to Section 2 of the supplementary materials to find the results for all datasets and hyperparameter search methods. When it comes to improving aggregate measure of

Table 6. Comparing performance of CPU and GPU using the 20 million synthetic data.

Test fold ID	# boosting rounds	Run time (sec)		
		CPU	GPU	Speedup
1	52	50.72	8.36	6.1×
2	149	150.33	22.49	6.7×
3	53	54.07	8.52	6.3×
4	81	81.33	12.44	6.5×
5	83	92.48	14.13	6.5×

generalization, test accuracy, it suffices to try 100 hyperparameter combinations; it does not make much difference in test accuracy to try more than 100 combinations.

3.4. Efficient Model Fitting with NVIDIA GPUs

XGBoost is able to use NVIDIA GPUs to accelerate its gradient boosting algorithm (Mitchell and Frank 2017; Ou 2020). We port the AFT loss function so that it can run on NVIDIA GPUs. To test the performance, we generate a synthetic dataset with 20 million samples, by duplicating 100,000 times⁵ the data `simulated.model.3` from Section 3.1.2. We then fit 5 XGBoost models using the five cross-validation folds. Each model is trained using the best hyperparameters found in Section 3.1.2. Table 6 shows the timing results. In all folds, the GPU fits the model 6.1–6.7× faster than the CPU.

We used NVIDIA Quadro® RTX 8000 with CUDA 10.2. The GPU has 4608 cores divided into 72 streaming multiprocessors and 48 GB GDDR6 memory.

4. Limitations

In this section we discuss two limitations of our current approach: sensitivity to hyperparameters and prediction of survival curves. First, even though Section 3.3 shows that the test accuracy is not very sensitive to the choice of hyperparameters, sensitivity to hyperparameters manifests itself in other ways. Vieira et al. (2021) present an example where two XGBoost AFT models that were fit with different values for the hyperparameter `aft_loss_distribution_scale` (and all the other hyperparameters the same) achieved nearly identical C-index metric on a validation dataset but produced highly different mean survival time on the same validation set. Aggregate metrics fail to account for this phenomenon.

In addition, the XGBoost software package lacks some tools that are often used in the literature of survival analysis, such as survival curve and confidence interval computation. The survival curve is defined as, for each time point t , the proportion of the population for which the event would complete by t . Although the XGBoost `predict` method only computes a point estimate (mean) of the survival time for each individual in the population, interested users could also compute a survival curve using the chosen AFT distribution and scale parameter. A concern with such survival curves is that they may not be

well-calibrated. In many applications, statistical models should be well calibrated to produce accurate probabilistic predictions, which in turn let us to accurately quantify the uncertainty around the given prediction.

It is possible to mitigate the aforementioned limitations. After our code became part of the XGBoost package on August 2020, a follow-up work (Vieira et al. 2021) addressed the shortcomings mentioned above, via model stacking. The authors created a new package XGBSE that built on top of the XGBoost AFT model, where the output of the XGBoost model is used as an input to a second model that is amenable to probability calibration, such as logistic regression or nearest neighbor. A survival curve is obtained by fitting a Kaplan–Meier estimator (Kaplan and Meier 1958) from the output of the second model. With this approach of model stacking, the authors were able to obtain well-calibrated survival curves that are not sensitive to the choice of hyperparameters. In short, XGBSE capitalized on existing strengths of XGBoost AFT while mitigating its limitations. The authors state that they chose to build on XGBoost AFT, because of its state-of-the-art discriminating power and generalization performance as given in test metrics.

5. Conclusion

We implemented the Accelerated Failure Time model in XGBoost, a widely used library for gradient boosting. Using real and simulated datasets, we show that AFT in XGBoost show competitive generalization performance and run-time efficiency, both for interval-censored and right-censored data. XGBoost gives superior generalization capacity compared to linear baselines, `survReg` and `penaltyLearning`, and runs faster than the nonlinear baseline, `mmit`. Furthermore, AFT in XGBoost is able to take advantage of many capabilities of the ecosystem of XGBoost, such as support for NVIDIA GPUs. A future work may take advantage of integration of XGBoost into distributed computing frameworks such as Apache Spark and Dask.

Since August 2020, when our work became part of the XGBoost package, it has enabled follow-up work in open-source statistical software. Already, packages such as XGBSE and MLR3 (Vieira et al. 2021; Lang et al. 2019) take advantage of the support for AFT in XGBoost. In particular, XGBSE was built on top of our work and addressed the major shortcomings (see Section 4). Usage of our software indicates real-world relevance and impact of our contribution.

Supplementary Materials

Supplementary_Material.pdf: Supplementary Material containing the full proof for Definition 3 and the full table for hyperparameter search results from Section 3.3.

JCGS_XGBoost_AFT_Tutorial.pdf: Short tutorial showing how to use AFT in XGBoost, in R and Python programming environments.

XGBoostAFTPaperCode/: Directory containing the full source code and scripts, needed to reproduce the experiments in the manuscript. Read `README.md` for instructions.

Acknowledgments

The author gratefully acknowledge Google for supporting him via Google Summer of Code 2019.

⁵The duplicated rows got the same fold assignment as their originals, so that the fraction of data points belonging to each cross-validation fold remains the same.

References

- Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019), “Optuna: A Next-Generation Hyperparameter Optimization Framework,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pp. 2623–2631, New York, NY: Association for Computing Machinery. <https://doi.org/10.1145/3292500.3330701>. [1298]
- Allison, P. D. (2010), *Survival Analysis Using SAS: A Practical Guide*, Cary, NC: SAS Institute. [1292]
- Barabadi, A., Barabady, J., and Markeset, T. (2010), “Application of Accelerated Failure Model for the Oil and Gas Industry in Arctic Region,” in *2010 IEEE International Conference on Industrial Engineering and Engineering Management*, pp. 2244–2248. [1293]
- Bühlmann, P., and Hothorn, T. (2007), “Boosting Algorithms: Regularization, Prediction and Model Fitting,” *Statistical Science*, 22, 477–505. <https://doi.org/10.1214/07-STS242>. [1293]
- Binder, H., and Schumacher, M. (2008), “Allowing for Mandatory Covariates in Boosting Estimation of Sparse High-Dimensional Survival Models,” *BMC Bioinformatics*, 9, 14. <https://doi.org/10.1186/1471-2105-9-14>. [1293]
- Bishop, C. M. (2006), *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Berlin: Springer-Verlag. [1294]
- Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984), *Classification and Regression Trees*, Boca Raton, FL: CRC press. [1296]
- Breslow, N. E. (1972), “Discussion on Professor Cox’s Paper,” *Journal of the Royal Statistical Society, Series B*, 34, 202–220. <https://doi.org/10.1111/j.2517-6161.1972.tb00900.x>. [1292]
- Buckley, J., and James, I. (1979), “Linear Regression with Censored Data,” *Biometrika*, 66, 429–436. [1292]
- Burges, C. J. C. (2010), “Fom RankNet to LambdaRank to LambdaMART: An overview,” Technical Report MSR-TR-2010-82. Available at <https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/>. [1293]
- Chen, T., and Guestrin, C. (2016), “XGBoost: A Scalable Tree Boosting System,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY: Association for Computing Machinery. <https://doi.org/10.1145/2939672.2939785>. [1293]
- Chen, T., Singh, S., Taskar, B., and Guestrin, C. (2015), “Efficient Second-Order Gradient Boosting for Conditional Random Fields,” in *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of Proceedings of Machine Learning Research, eds. G. Lebanon and S. V. N. Vishwanathan, pp. 147–155, San Diego, CA: PMLR. Available at <http://proceedings.mlr.press/v38/chen15b.html>. [1293]
- Chiou, S., Kang, S., and Yan, J. (2014), “Fitting Accelerated Failure Time Models in Routine Survival Analysis with R package aftgee,” *Journal of Statistical Software*, 61, 1–23. <https://doi.org/10.18637/jss.v061.i11>. [1292]
- Cox, D. R. (1972), “Regression Models and Life-Tables,” *Journal of the Royal Statistical Society, Series B*, 34, 187–220. [1292]
- Dirick, L., Claeskens, G., and Baesens, B. (2017), “Time to Default in Credit Scoring Using Survival Analysis: A Benchmark Study,” *Journal of the Operational Research Society*, 68, 652–665. <https://doi.org/10.1057/s41274-016-0128-9>. [1293]
- Drouin, A., Hocking, T., and Laviolette, F. (2017), “Maximum Margin Interval Trees,” in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, pp. 4947–4956. Curran Associates, Inc., 2017. Available at <http://papers.nips.cc/paper/7080-maximum-margin-interval-trees.pdf>. [1293,1296,1297]
- Faruk, A. (2018), “The Comparison of Proportional Hazards and Accelerated Failure Time Models in Analyzing the First Birth Interval Survival Data,” *Journal of Physics: Conference Series*, 974, 012008. <https://doi.org/10.1088%2F1742-6596%2F974%2F1%2F012008>. [1292]
- Friedman, J., Hastie, T., and Tibshirani, R. (2000), “Additive Logistic Regression: A Statistical View of Boosting,” (with Discussion and a Rejoinder by the Authors), *The Annals of Statistics*, 28, 337–407. <https://doi.org/10.1214/aos/1016218223>. [1293]
- Friedman, J. H. (2001), “Greedy Function Approximation: A Gradient Boosting Machine,” *The Annals of Statistics*, 29, 1189–1232. [1292]
- Harrell, F. E., Jr., Califf, R. M., Pryor, D. B., Lee, K. L., and Rosati, R. A. (1982), “Evaluating the Yield of Medical Tests,” *JAMA: The Journal of the American Medical Association*, 247, 2543–2546. <https://doi.org/10.1001/jama.1982.03320430047030>. [1298]
- Hocking, T. D., Goerner-Potvin, P., Morin, A., Shao, X., Pastinen, T., and Bourque, G. (2016), “Optimizing ChIP-seq Peak Detectors Using Visual Labels and Supervised Machine Learning,” *Bioinformatics*, 33, 491–499. <https://doi.org/10.1093/bioinformatics/btw672>. [1296]
- Hothorn, T., Lausen, B., Benner, A., and Radespiel-Tröger, M. (2004), “Bagging Survival Trees,” *Statistics in Medicine*, 23, 77–91. <https://doi.org/10.1002/sim.1593>. [1293]
- IEEE (2019), “IEEE Standard for Floating-Point Arithmetic,” *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84. [1295]
- Ishwaran, H., Kogalur, U. B., Blackstone, E. H., and Lauer, M. S. (2008), “Random Survival Forests,” *The Annals of Applied Statistics*, 2, 841–860. <https://doi.org/10.1214/08-AOAS169>. [1293]
- Kaplan, E. L., and Meier, P. (1958), “Nonparametric Estimation from Incomplete Observations,” *Journal of the American Statistical Association*, 53, 457–481. [1300]
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017), “LightGBM: A Highly Efficient Gradient Boosting Decision Tree,” in *Advances in Neural Information Processing Systems 30*, eds. I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, pp. 3146–3154. Curran Associates, Inc. <http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf>. [1293]
- Khan, M. H., and Shaw, J. E. H. (2016), “Variable Selection for Survival Data with a Class of Adaptive Elastic Net Techniques,” *Statistics and Computing*, 26, 725–741. <https://doi.org/10.1007/s11222-015-9555-8>. [1292]
- Kvamme, H., Borgan, Ø., and Scheel, I. (2019), “Time-to-Event Prediction with Neural Networks and Cox Regression,” *Journal of Machine Learning Research*, 20, 1–30. [1293]
- Lang, M., Binder, M., Richter, J., Schratz, P., Pfisterer, F., Coors, S., Au, Q., Casalicchio, G., Kotthoff, L., and Bischl, B. (2019), “mlr3: A Modern Object-Oriented Machine Learning Framework in R,” *Journal of Open Source Software*, 4, 1903. <https://doi.org/10.21105/joss.01903> [1300]
- Lundberg, S. M., Erion, G. G., and Lee, S.-I. (2019), “Consistent Individualized Feature Attribution for Tree Ensembles.” <https://arxiv.org/abs/1802.03888> [1293,1298]
- Miller, R. G. (1976), “Least Squares Regression with Censored Data,” *Biometrika*, 63, 449–464. [1292]
- Mimi, A., and Khan, M. H. R. (2021), “Variable Selection for Censored Data Using Modified Correlation Adjusted Correlation (mcar) Scores,” *Statistics in Medicine*, 40, 5046–5064. <https://doi.org/10.1002/sim.9110> [1292]
- Mitchell, R., and Frank, E. (2017), “Accelerating the XGBoost Algorithm Using GPU Computing,” *PeerJ Computer Science*, 3, e127. [1300]
- Ou, R. (2020), “Out-of-Core GPU Gradient Bboosting.” <https://arxiv.org/abs/1802.03888> [1300]
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. (2011), “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, 12, 2825–2830. [1293]
- Pösterl, S. (2020), “scikit-survival: A Library for Time-to-Event Analysis Built on Top of scikit-learn,” *Journal of Machine Learning Research*, 21, 1–6. [1293,1298]
- Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. (2018), “CatBoost: Unbiased Boosting with Categorical Features,” in *Advances in Neural Information Processing Systems 31*, eds. S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, pp. 6638–6648. Curran Associates, Inc. Available at <http://papers.nips.cc/paper/7898-catboost-unbiased-boosting-with-categorical-features.pdf>. [1293]
- Rigaill, G., Hocking, T., Vert, J.-P., and Bach, F. (2013), “Learning Sparse Penalties for Change-Point Detection Using Max Margin Interval Regression,” in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pp. 172–180. [1296]

- Schmid, M., and Hothorn, T. (2008), "Flexible Boosting of Accelerated Failure Time Models," *BMC Bioinformatics*, 9, 269–269. [1293]
- Susto, G. A., Schirru, A., Pampuri, S., McLoone, S., and Beghi, A. (2015), "Machine Learning for Predictive Maintenance: A Multiple Classifier Approach," *IEEE Transactions on Industrial Informatics*, 11, 812–820. [1293]
- Therneau, T. M. (2015), "A Package for Survival Analysis in S," version 2.38. Available at <https://CRAN.R-project.org/package=survival>. [1294,1295]
- Uno, H., Cai, T., Pencina, M. J., D'Agostino, R. B., and Wei, L.-J. (2011), "On the C-statistics for Evaluating Overall Adequacy of Risk Prediction Procedures with Censored Survival Data," *Statistics in Medicine*, 30, 1105–1117. <https://doi.org/10.1002/sim.4154>. [1298]
- Vaid, A., Somani, S., Russak, A. J., De Freitas, J. K., Chaudhry, F. F., Paranjpe, I., Johnson, K. W., Lee, S. J., Miotto, R., Richter, F., Zhao, S., Beckmann, N. D., Naik, N., Kia, A., Timsina, P., Lala, A., Paranjpe, M., Golden, E., Danieletto, M., Singh, M., Meyer, D., O'Reilly, P. F., Huckins, L., Kovatch, P., Finkelstein, J., Freeman, R. M., Argulian, E., Kasarskis, A., Percha, B., Aberg, J. A., Bagiella, E., Horowitz, C. R., Murphy, B., Nestler, E. J., Schadt, E. R., Cho, J. H., Cordon-Cardo, C., Fuster, V., Charney, D. S., Reich, D. L., Bottinger, E. P., Levin, M. A., Narula, J., Fayad, Z. A., Just, A. C., Charney, A. W., Nadkarni, G. N., and Glicksberg, B. S. (2020), "Machine Learning to Predict Mortality and Critical Events in a Cohort of Patients with COVID-19 in New York City: Model Development and Validation," *Journal of Medical Internet Research*, 22, e24018. <https://doi.org/10.2196/24018>. [1293]
- Van den Poel, D., and Larivière, B. (2004), "Customer Attrition Analysis for Financial Services Using Proportional Hazard Models," *European Journal of Operational Research*, 157, 196–217. [https://doi.org/10.1016/S0377-2217\(03\)00069-9](https://doi.org/10.1016/S0377-2217(03)00069-9). <http://www.sciencedirect.com/science/article/pii/S0377221703000699>. Smooth and Nonsmooth Optimization. [1293]
- Vieira, D., Gimenez, G., Marmerola, G., and Estima, V. (2021), "XGBoost Survival Embeddings: Improving Statistical Properties of XGBoost Survival Analysis Implementation," Available at <http://github.com/loft-br/xgboost-survival-embeddings>. [1300]
- Viganò, A., Dorgan, M., Buckingham, J., Bruera, E., and Suarez-Almazor, M. E. (2000), "Survival Prediction in Terminal Cancer Patients: A Systematic Review of the Medical Literature," *Palliative Medicine*, 14, 363–374. <https://doi.org/10.1191/026921600701536192>. [1293]
- Wang, P., Li, Y., and Reddy, C. K. (2019), "Machine Learning for Survival Analysis: A Survey," *ACM Computing Surveys*, 51, 1–36. <https://doi.org/10.1145/3214306>. [1292]
- Wei, L.-J. (1992), "The Accelerated Failure Time Model: A Useful Alternative to the Cox Regression Model in Survival Analysis," *Statistics in Medicine*, 11, 1871–1879. <https://doi.org/10.1002/sim.4780111409>. [1292]
- Zhang, T., and Yu, B. (2003), "On the Convergence of Boosting Procedures," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pp. 904–911, AAAI Press. [1293]