

# BART-Survival: A Bayesian machine learning approach to survival analyses in Python

Jacob Tiegs<sup>1,2</sup>, Julia Raykin<sup>1</sup>, and Ilia Rochlin<sup>1</sup>

<sup>1</sup> Inform and Disseminate Division, Office of Public Health Data, Surveillance, and Technology, Centers for Disease Control and Prevention, Atlanta, GA, USA <sup>2</sup> Metas Solutions, Atlanta, Georgia ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- Review
- Repository
- Archive

Editor: [Open Journals](#)

## Reviewers:

- @openjournals

Submitted: 01 January 1970

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## Summary

BART-Survival is a Python package that allows time-to-event (survival) analyses in discrete-time using the non-parametric machine learning algorithm, Bayesian Additive Regression Trees (BART). BART-Survival combines the performance of the BART algorithm with the complementary data and model structural formatting required to complete the survival analyses. The library contains a convenient application programming interface (API) that allows a simple approach when using the library for survival analyses, while maintaining capabilities for added complexity when desired. The package is intended for analysts exploring use of flexible non-parametric alternatives to traditional (semi-)parametric survival analyses.

## Statement of need

Survival analyses are a cornerstone of public health and clinical research in such diverse fields as cancer, cardiovascular disease, and infectious diseases (Altman & Bland, 1998; Bradburn et al., 2003). Traditional parametric and semi-parametric statistical methods, such as the Cox proportional hazards model, are commonly employed for survival analyses (Cox, 1972). However, these methods have several limitations, particularly when applied to complex data. One major issue is the need for restrictive assumptions, such as proportional hazards and predefined functional forms, which may not hold true in complex, real-world healthcare data (Harrell, 2015; Ishwaran et al., 2008). Additionally, these methods often struggle with high-dimensional datasets, leading to problems with overfitting, multicollinearity, and dealing with complex interactions (Ishwaran et al., 2008; Joffe et al., 2013).

More recently, non-parametric machine learning approaches have been introduced to address these limitations by reducing the need for restrictive assumptions and providing increased capabilities for more accurately modeling underlying distributions and complex interactions (Harrell, 2015; Ishwaran et al., 2008). BART is one such machine learning method that has demonstrated utility in the survival setting through its performance in identifying underlying statistical distributions (Chipman et al., 2010; R. Sparapani et al., 2021). BART offers flexibility in modeling complex relationships and interactions within the data without requiring the specification of a particular functional form (R. A. Sparapani et al., 2016).

Currently, the only BART survival algorithm readily available exists as part of the BART R package, which contains a library of various BART-based approaches in addition to a BART survival analysis application (R. Sparapani et al., 2021; R. A. Sparapani et al., 2016). The BART-Survival package described here combines the survival analysis approach outlined in the BART R package with the foundational Python-based probabilistic programming language library, PyMC (Abril-Pla et al., 2023), and the accompanying BART algorithm from the PyMC-BART library (Quiroga et al., 2023). Our aim in developing BART-Survival is to provide accessibility

to the BART survival algorithm within the Python programming language. This contribution is beneficial for analysts when Python is the preferred programming language, the analytic workflow is Python-based, or when the R language is unavailable for analyses.

The need for a complete BART-Survival python package is given by the simple fact that the BART survival algorithm is non-trivial to implement. Both the required data transformations and the internal model definition requires precise implementations to ensure generation of accurate survival models. Our BART-Survival library provides accessibility to these precise methods while removing the technical barriers that would limit user adoption of the BART survival approach.

More specifically, the BART-Survival library abstracts away the complexities of generating the proper training and inference datasets, which are conceptually complex and prone to being specified incorrectly if implemented from scratch. Similarly, the BART-Survival library provides a pre-specified internal Bayesian model using the PyMC probabilistic programming language. This pre-specified model is primarily accessed through the BART-Survival API removing the requirement for users to have more than a cursory knowledge of the PyMC or PyMC-BART libraries. Since the BART-Survival package is intended for students and professional in the public health and clinical fields, it is expected users of the BART-Survival library will not have extensive programming expertise, adding to the need for a full contained and accessible approach.

In summary the BART-Survival package provides a simple and accessible approach to implementing the BART survival algorithm. The provided approach can be beneficial for users who are looking for non-parametric alternatives to traditional (semi-)parametric survival analysis. The BART survival algorithm can be especially useful in large, complex healthcare data, where machine learning methods can demonstrate improved performance over the traditional methods.

## Methods

The following sections provides details on the methods employed by the BART-Survival library, focusing specifically on the discrete-time survival algorithm used. For review of the BART algorithm we refer to associated PyMC-BART publication (Quiroga et al., 2023).

## Background

The BART-Survival package provides a discrete-time survival method which aims to model survival as a function of a series of probabilities that indicate the probability of event occurrence at each discrete time. For clarity, the probability of event occurrence at each discrete time will be referred to as the risk probability.

In combination with a structural configuration of the data, the discrete-time algorithm allows for flexible modeling of the risk probabilities as a non-parametric function of time and observed covariates. The series of probability risks can then be used to derive the survival probabilities, along with other estimates of interest.

The foundation of the method is simple and is based off the well-defined Kaplan-Meier Product-Limit estimator. While a full review of the Kaplan-Meier method can be found elsewhere (Stel Vianda S., 2011), the following example demonstrates the fundamental concepts of discrete-time survival analysis that motivates its application within the BART-Survival method.

1. Starting with a simple event-time dataset, create a sequence of time intervals that represent the unique discrete-time intervals observed in the data. Each interval is represented as a  $t_j$ , where  $j = 1, \dots, k$  and  $k$  is the length of the set of unique observed times.

For example if the observed event-time data is:

event status	event time
1	1
0	2
1	2
1	4
1	4
1	5

88 Then the set of unique observed times is  $[1, 2, 4, 5]$  with  $k = 4$  indices and the corre-  
89 sponding  $t_j$  intervals are:

index	interval
$t_1$	$(0, 1]$
$t_2$	$(1, 2]$
$t_3$	$(2, 4]$
$t_4$	$(4, 5]$

90 In the table above the intervals are denoted with the properties, "(" indicating exclusive  
91 times and "]" indicating inclusive times in an interval. Additionally, the **event status**  
92 column is defined as (1 = event, 0 = censor)

93 2. Then with the constructed time intervals  $t_1, \dots, t_k$ , tally the following for each interval:

- 94 ■ the number of observations with an event
- 95 ■ the number of observation censored
- 96 ■ the total number of observations eligible to have an event (at risk) at the start of
- 97 the interval

98 Continuing the above example the corresponding frequencies for each interval are:

index	interval	event	censor	at risk
$t_1$	$(0, 1]$	1		6
$t_2$	$(1, 2]$	1	1	5
$t_3$	$(2, 4]$	2		3
$t_4$	$(4, 5]$	1		1

99 3. Finally, the risk of event occurrence within each interval  $t_j$  can simply be derived as:

$$P_{t_j} = \frac{n \text{ events}_{t_j}}{n \text{ at risk}_{t_j}},$$

100 and the survival probability at a time-index  $j$ , can be derived as:

$$S(t_j) = \prod_{q=1}^j (1 - P_{t_q}).$$

101 Applied to our example the risks of event  $P_{t_j}$  are:

index	interval	event	censor	at risk	$P_{t_j}$
$t_1$	$(0, 1]$	1		6	0.167
$t_2$	$(1, 2]$	1	1	5	0.2
$t_3$	$(2, 4]$	2		3	0.667
$t_4$	$(4, 5]$	1		1	1.0

102 And the corresponding survival estimates at times  $[1, 2, 4, 5]$  are:

index	time	survival
$t_1$	1	.83
$t_2$	2	.67
$t_3$	4	.22
$t_4$	5	0

103 BART-Survival builds off this simple foundation by replacing each naively derived  $P_{t_j}$  with a  
 104 probability risk prediction,  $p_{t_j|x_{ij}}$  yielded from the BART regression model. The  $x_{ij}$  values are  
 105 the associated covariate values for each observation  $i$  at each discrete time  $j$ . The predicted  
 106 values  $p_{t_j|x_{ij}}$  can be further transformed into survival probability estimates or other estimands  
 107 of interest. Formally, the survival probability estimate for a single observation  $i$  at time-index  $j$   
 108 can be derived as:

$$S(t_j|x_i) = \prod_{q=1}^j (1 - p_{t_q|x_{iq}}).$$

## 109 Data Preparation

110 As displayed in the above example, survival data is typically given as paired (*event status*,  
 111 *event time*) outcomes, along with a set of covariates  $x_i$  for each observation. In this setup,  
 112 *event status* is typically a binary variable, with 1 = **event** and 0 = **censored**.

113 Obtaining the  $p_{t_j|x_{ij}}$  values and subsequent survival estimates is a two-step process. Each step  
 114 of the process requires generating a specialized augmented datasets from the generic survival  
 115 data. The first step in the process involves training the BART model which requires the  
 116 **training augmented dataset (TAD)**. The second step involves generating the  $p_{t_j|x_{ij}}$  predictions  
 117 from the trained BART model, which requires the **predictive augmented dataset (PAD)**.

118 The **TAD** is created by transforming each single observation row into a series of rows representing  
 119 the observation over a time series. When transformed a single observation's information is  
 120 represented as a series of tuples, each tuple containing the augmented values of *status* and  
 121 *time*, as well as replicates of the covariates  $x$ . The series of tuples is constructed using the  
 122  $k$  distinct times from the generic dataset. These times are represented by  $t$ , which can be  
 123 indexed by  $j$ , where  $j$  is in the set  $1, \dots, k$ . Then for each observation's *event time* <sub>$i$</sub>  there is an  
 124 index  $j$  in  $1, \dots, k$  where  $t_j = \text{event time}_i$ . This special index  $j$  is denoted as  $m_i$  and can be  
 125 defined as:

$$t_{m_i} = t_j = \text{event time}_i.$$

126 For each observation, a set of tuples is created containing a total of  $m_i$  tuples which are used  
 127 to represent that observation's information. Each tuple is created for a specific time  $t_j$  where  
 128  $j = 1, \dots, m_i$ . Additionally, for all tuples with times  $t_j < t_{m_i}$  the value of *status* is set to 0

129 and for the time  $t_j = t_{m_i}$  the value of *status* is set to the value *event status<sub>i</sub>*. Covariate  
130 information is simply replicated across tuples.

131 For example if the survival data is:

observation	status	time	unique times (k=6)
— — —	— — —	— — —	
1	1	4	with 4 6 7 8 12 14
2	0	6	
3	1	6	
4	1	7	
5	1	8	
6	0	8	
7	1	12	
8	1	14	

132 Then the transformation for observation 7 would be represented in the *augmented* dataset as  
133 the sequence of rows:

observation	status	time
— — —	— — —	— — —
7	0	4
7	0	6
7	0	7
7	0	8
7	1	12

134 Similarly the transformation for observation 6 would be represented as:

observation	status	time
— — —	— — —	— — —
6	0	4
6	0	6
6	0	7
6	0	8

135 It is important to reiterate that for each observation  $i$ , the new set of rows created only include  
136 the time points  $t_j \leq t_{m_i}$ . For observation 7 the  $t_{m_i} = 12$  and the corresponding times in  
137 the **TAD** are 4, 6, 7, 8, 12. For observation 6 the  $t_{m_i} = 8$  and the corresponding times in the  
138 **TAD** are 4, 6, 7, 8.

139 The utility of the **TAD** is that it unlinks *event time* from the *event status*. In this setup, the  
140 constructed *status* variable (which is a simple binary variable taking values 1 or 0) represents  
141 the outcome and *time* is a covariate. Then treating each row of the **TAD** as an independent  
142 observation, the outcome can be modeled as a binary regression of *status* over *time* and any  
143 additional covariates  $x$ . The trained regression model from this dataset can be used to yield  
144 probability predictions for each time  $t_j$  for  $j$  in  $1, \dots, k$ . These probability predictions hold the  
145 interpretation as being the probability risk of event occurrence at each time conditional on the  
146 event not having already occurred at a previous time. The predictions for each observation  $i$   
147 and time  $t_j$  can be used to yield the various analytic targets, such as survival probabilities.

148 To generate the predictions of probability risks over the observed times the **PAD** dataset is used.  
149 The **PAD** transformation is similar to the **TAD**, but is simpler in that for each observation a  
150 set of size  $k$  tuples is created. Each tuple in the set contains the time  $t_j$  for  $j = 1, \dots, k$  and  
151 any additional covariates  $x$ .

152 Continuing the examples above, there are  $k = 6$  the unique times. For  $j = 1, \dots, k$  the resulting  
153 distinct times are  $t_j = 4, 6, 7, 8, 12, 14$  and the **PADs** for observations 6, and 7 would be:

observation	time
6	4
6	6
6	7
6	8
6	12
6	14
7	4
7	6
7	7
7	8
7	12
7	14

154 Notably the **TAD** and **PAD** lengths can differ. **PAD** length is simple to calculate. If  $n$  is  
 155 the number of observations in the generic dataset and  $k$  is the number of the unique times  
 156 observed in the generic dataset, then the length of the **PAD** is simply  $n * k$ . In the example  
 157 data above it would be  $(8 * 6) = 48$ .

158 The **TAD** length can be calculated using each observation's  $m_i$ , which again is defined as  
 159 being the index that resolves  $t_{m_i} = \text{event time}_i$ . The **TAD** length can be calculated as:

$$\text{TAD}_{\text{length}} = \sum_{i=1}^n m_i.$$

160 This can be demonstrated with the example dataset where the **TAD** length can be found as  
 161 the sum of the  $m_i$  column which resolves to a **TAD** length of 27:

observation	event time <sub>i</sub>	$m_i$
1	4	1
2	6	2
3	6	2
4	7	3
5	8	4
6	8	4
7	12	5
8	14	6
		27

162 While the **PAD** and **TAD** lengths are not important for generating the models, they are helpful  
 163 to keep in mind when completing an analysis. Specifically, the fact that the **TAD** and **PAD**  
 164 can be far larger in length than the generic dataset. For example the generic dataset used  
 165 above is only 8 rows in length, but the **TAD** and **PAD** dataset are 27 and 48 rows respectively.

166 When the length of the two datasets become large enough to make computation difficult it is  
 167 recommended to downscale the *event time* values, which will reduce the length of the two  
 168 augmented datasets. The downscaling algorithm we provide in the library takes the *event time*,  
 169 divides by a scaling factor and then takes the "ceiling" truncation to create the new *scaled*  
 170 *event time*. Below the downscaling algorithm is demonstrated with the example dataset using  
 171 a scaling factor of 4:

time	downscaled time	rescaled time
4	1	4
6	2	8
6	2	8
7	2	8
8	2	8
8	2	8
12	3	12
14	4	16

Downscaling the *event time* values causes loss of granularity in the set of discrete times analyzed. As shown in the example above, after downscaling (and then rescaling) the analytic targets can only be derived for the times 4, 8, 12 and 16, which is reduced from the original set of times, 4, 6, 7, 8, 12, 14. While the analytic targets cannot be returned for the times 6, 7, 14, the training information contributed by the observations with these *event times* is maintained through subsequent contribution at the downscaled times. Since there is no loss of event information, the probability of event at the downscaled times 4, 8, 12, 16 will be equal to the probability of event from a non-downscaled dataset at the select time 4, 8, 12, 16.

Downscaling can be safely applied to datasets without loss of precision of estimates at the downscaled times. The only consideration required when using a downscaling procedure is to ensure that the downscaled times granularity fulfills the needs of the analysis. For example, in a multi-year health-outcomes study, downscaling from days to months could significantly reduce computational burden without a meaningful loss of information in the outcome. However, if that same study reduced time from days to years, this could lead to loss of meaningful information in the outcome. The considerations for downscaling need to be made on a study-to-study basis and no general recommendation on downscaling factor can be provided.

## Model

The BART-Survival algorithm is two-step algorithm. First, the **TAD** is used to train a non-parametric regression model of *status* on *time* and covariates. Then the **PAD** is used to yield predicted probabilities at each discrete time. These probability of event predictions can be used generate all additional estimates useful for statistical inference.

To motivate the use of the **TAD** in training the regression model an example **TAD** is displayed below. Here *status* can be represented as the outcome vector of  $y_{ij}$  values and *time* is represented by the  $t_{ij}$  values. Additional covariates are represented by a vector of  $x_{ij}$  values. The subscript  $i$  refers to the  $i^{th}$  observation of the generic dataset and the subscript  $j$  refers to the  $j^{th}$  index of the unique set of discrete times previously defined as  $t$ . In the **TAD** and **PAD**, the  $i, j$  indices can be thought of as multi-indices over the rows.

$i$	$j$	$y$	$t$	$x$
1	1	0	3	$x_{11}$
1	2	0	4	$x_{12}$
1	3	1	6	$x_{13}$
2	1	0	3	$x_{21}$
2	2	1	4	$x_{22}$
3	1	0	3	$x_{31}$
3	2	0	4	$x_{32}$
3	3	1	6	$x_{33}$

When using the **TAD**, each  $y_{ij}$  value can then be treated as independent draw of a *Bernoulli*



distribution parameterized with the probability of event  $p_{ij}$  for observation  $i$  at time index  $j$ . The  $p_{ij}$  values are collected as the latent values yielded from a *probit* regression of  $y$  on  $t$  and  $x$ . Formally the model is defined as:

$$\begin{aligned} y_{ij}|p_{ij} &\sim \text{Bernoulli}(p_{ij}), \\ p_{ij}|\mu_{ij} &= \Phi(\mu_{ij}), \\ \mu_{ij} &\sim \text{BART}(t_{ij}, x_{ij}), \end{aligned}$$

where  $\Phi$  is the normal cumulative distribution function and BART is the ensemble of regression trees which yield the  $\mu_{ij}$  value for a given  $t_{ij}$ ,  $x_{ij}$  combination.

Regarding the BART algorithm, in brief BART is a Bayesian approach to the ensemble regression trees class of non-parametric models. BART uses a combination of Bayesian priors placed on the tree generating components and a Markov Chain Monte Carlo sampling algorithm to iteratively generate tree ensembles which are collected as samples of the posterior distribution. The samples of the posterior distribution are subsequently used to generate distinct posterior predictive distributions for a given set of observations.

This implies that for each  $t_{ij}$ ,  $x_{ij}$  combination, the algorithm first returns a distribution of  $p_{ij}$  values, denoted as  $p_{ij}^{dist}$ . Each value in the  $p_{ij}^{dist}$  is a prediction mapped from a single tree ensemble of the posterior distribution. Point estimates and uncertainty intervals can be easily obtained from the  $p_{ij}^{dist}$  as simple estimates from that distribution. For example the mean can be derived as the empirical average of  $p_{ij}^{dist}$ . Similarly, the even-tailed 95% credible interval can be derived as the 5<sup>th</sup> and 95<sup>th</sup> percentiles of the  $p_{ij}^{dist}$ . Of note any reference to  $p_{ij}$  used in this article is referring to the mean point estimate of the  $p_{ij}^{dist}$ .

Further details regarding the BART implementation used in the BART-Survival algorithm can be found in PyMC-BART repository and the accompanying publication (Quiroga et al., 2023).

## Survival

A trained BART-Survival model is used along with the **PAD** to generate the vector of  $p_{ij}$  predictions. To generate survival estimates the  $p_{ij}$  vector is grouped by the  $i$  indices. Then using the following equation the survival probability at discrete times can be constructed:

$$S(t_q|x_{iq}) = \prod_{j=1}^q (1 - p_{ij}),$$

where  $q$  is the index of the time for the desired estimate. As described in the previous section a full posterior predictive distribution is returned for each  $S_{t_q|x_{iq}}$ . Point estimates and credible intervals can be obtained through the empirical mean and percentile functions of the returned posterior predictive distribution for each  $S(t_q|x_{iq})$ .

## Marginal Effects

BART-Survival provides capabilities to evaluate covariate effects through use of partial dependence functions which yield marginal effect estimates. The partial dependence function method involves generating predictions of  $p$  for the observations in a **partial dependence augmented dataset (PDAD)**. Using variations of the **PDAD** yields different sets of predictions  $p$ . These sets can then be contrasted to yield marginal effect estimates, including marginal Hazard Ratios which have similar interpretation as the Cox Proportional Hazard Model's conditional Hazard Ratios.

**PDADs** can be created through further augmentation of the previously described **PAD**. As a reminder, the **PAD** contains the generated time covariate  $t$  and  $k$  replicates of each  $x_i$  from the generic dataset, where  $k$  is the length of the uniquely observed *event times*. Importantly



$x_i$  can be defined generically as the set of covariates  $\{x_1, x_2, \dots, x_m\}$ , where  $m$  is the index of the last included covariate.

Then, starting with the **PAD**, the **PDAD** is generated through selection of a specific covariate  $x_{[I]}$ , from the set of covariates  $\{x_1, x_2, \dots, x_m\}$  and setting that covariate to a deterministic value for all observation  $i$  and evaluated times  $j$ . The bracketed notation  $[I]$  is used to define the index of the covariate selected and convey that the selected covariate is set to a deterministic value. Expanding the notation,  $x_{[I]=v}$  indicates that covariate of index  $I$  is selected and deterministically set to value  $v$ .

In the following example the second covariate from a set of covariates  $\{x_1, x_2, x_3\}$  is of interest for evaluating marginal effects. This covariate is binary and takes the values 0, 1. Evaluation of this covariate can be completed by creating **PDADs** for possible values of the selected covariate. In each **PDAD** all of the observations and times are set to a deterministic values selected from the range of the covariate. The two generated **PDADs** can be defined by the covariate index and set values:  $x_{[2]=0}$  and  $x_{[2]=1}$ .

$i$	$j$	$t$	$x_1$	$x_2$	$x_3$
1	1	2	1.2	0	10
1	2	3	1.2	0	10
1	3	5	1.2	0	10
PAD: 2					
2	1	2	2.4	1	12
2	2	3	2.4	1	12
2	3	5	2.4	1	12
3	1	2	1.9	0	3
3	2	3	1.9	0	3
3	3	5	1.9	0	3

$i$	$j$	$t$	$x_1$	$x_2$	$x_3$
1	1	2	1.2	1	10
1	2	3	1.2	1	10
1	3	5	1.2	1	10
PDAD $x_{[2]=1}$					
2	1	2	2.4	1	12
2	2	3	2.4	1	12
2	3	5	2.4	1	12
3	1	2	1.9	1	3
3	2	3	1.9	1	3
3	3	5	1.9	1	3

$i$	$j$	$t$	$x_1$	$x_2$	$x_3$
1	1	2	1.2	0	10
1	2	3	1.2	0	10
1	3	5	1.2	0	10
PDAD $x_{[2]=0}$					
2	1	2	2.4	0	12
2	2	3	2.4	0	12
2	3	5	2.4	0	12
3	1	2	1.9	0	3
3	2	3	1.9	0	3
3	3	5	1.9	0	3

From the two **PDADs** the predicted probabilities  $p_{x_{[2]=1}}$ ,  $p_{x_{[2]=0}}$  and survival probabilities  $S_{x_{[2]=1}}$ ,  $S_{x_{[2]=0}}$  can be generated:

$i$	$j$	$p_{x_{[2]=1}}$	$S_{x_{[2]=1}}$	$p_{x_{[2]=0}}$	$S_{x_{[2]=0}}$
1	1	.20	.80	.10	.90
1	2	.25	.60	.15	.77
1	3	.18	.49	.08	.70
2	1	.10	.90	.08	.92
2	2	.13	.78	.11	.81
2	3	.12	.69	.10	.74
3	1	.20	.80	.15	.92
3	2	.28	.58	.23	.82
3	3	.23	.44	.18	.74

The marginal expectations of  $p_{x_{[t]}}$  and  $S_{x_{[t]}}$  at a specific time can be further derived by taking the average of the estimates over observations  $i$  for the specified time  $t$  indexed by  $j$ :

$$E_i[p_{x_{[t]}}|t_j] = \frac{1}{n} \sum_{i=1}^n p_{ij,x_{[t]}},$$

$$E_i[S_{x_{[t]}}(t_j)] = \frac{1}{n} \sum_{i=1}^n S_{i,x_{[t]}}(t_j),$$

where  $E_i$  is the expectation over  $i, \dots, n$  observations. From the above example this yields the expectations for time indices  $j = 1, 2, 3$ .

$j$	$t$	$E_i[p_{x_2=1}]$	$E_i[S_{x_2=1}]$	$E_i[p_{x_2=0}]$	$E_i[S_{x_2=0}]$
1	2	0.17	0.83	0.11	0.91
2	3	0.22	0.65	0.16	0.8
3	5	0.18	0.54	0.12	0.73

These expectations can be further used to make comparisons between the evaluation of various values of  $x_{[t]}$  across multiple PDADs. Common marginal effect estimates derived from these predicted values include:

- Marginal difference is survival probability at time  $t$ :

$$\text{Surv. Diff.}_{\text{marg}}(t_j) = E_i[S_{x_{[t]=v_2}}(t_j)] - E_i[S_{x_{[t]=v_1}}(t_j)].$$

- Marginal Risk Ratio at time  $t$ :

$$\text{RR}_{\text{marg}}(t_j) = \frac{E_i[p_{j,x_{[t]=v_2}}]}{E_i[p_{j,x_{[t]=v_1}}]}.$$

- Marginal Hazard Ratio (expectation over  $i$  and times up to  $t$ ):

$$\text{HR}_{\text{marg}}(t_j) = \frac{E_{ij}[p_{x_{[t]=v_2}}]}{E_{ij}[p_{x_{[t]=v_1}}]}.$$

Continuing the example, the marginal effect of  $x_2$  as measured by difference in survival probability when  $x_{[2]} = 1$  and  $x_{[2]} = 0$  at the times 2, 3, 5 can be examined below:

$j$	$t$	Surv.Diff.
1	2	−0.08
2	3	−0.15
3	5	−0.19

These results can be conveniently interpreted. For example, at time  $t = 5$  the increase of  $x_2$  from 0 to 1 leads to an average change in the survival probability of  $-0.19$ , where survival probability is in the range 0-1. Additionally, as mentioned in the previous sections, all estimates of the model will first be yielded as posterior predictive distributions. The empirical mean and percentile function of this distribution yield the point estimates described above and their respective credible intervals.

## Demonstration

The following is a brief demonstration on how to use the BART-Survival library. In the demonstration the *rossi* survival dataset from the lifelines library is used. The dataset contains one year of follow-up observation on 432 convicts who were released from Maryland state prisons in the 1970s. The primary event measured in this dataset is observation of a “new arrest” within that year and the associated “time to arrest” is given in weeks (Rossi & Lenihan, 1980).

```
from lifelines.datasets import load_rossi
from bart_survival import surv_bart as sb
import numpy as np
```

```
#####
# Load rossi dataset from lifelines
rossi = load_rossi()
names = rossi.columns.to_numpy()
rossi = rossi.to_numpy()
```

After loading the libraries and data, the first step is to generate the **TAD** and **PAD** datasets. In this step, the time (originally in weeks) is downsampled by a factor of 4, setting time to be measured in months.

```
#####
# Transform data into 'augmented' dataset
# Requires creation of the training dataset and a predictive dataset for inference
# TAD
trn = sb.get_surv_pre_train(
    y_time=rossi[:,0],
    y_status=rossi[:,1],
    x = rossi[:,2:],
    time_scale=4
)

# PAD
post_test = sb.get_posterior_test(
    y_time=rossi[:,0],
    y_status=rossi[:,1],
    x = rossi[:,2:],
    time_scale=4
)
```

Below the *trn* object is displayed. The *trn* object generated in this step is a dictionary of arrays containing the **TAD** components. The *y* and *x* components are the corresponding outcome and covariates of the **TAD**. The *w* object is an array of weight values generated by the *get\_surv\_pre\_train* function. By default all weights are set to 1 and do not contribute to model training. For general use the weighting functionality can be ignored. In more complex study designs, observation level weights can be provided which allows weighted contribution to the likelihood function during model training. Weighting the likelihood function is currently

experimental, but we plan to evaluate this utility further in future work. Finally, the `coord` object contains the observation identifier for each row of the **TAD** making it easy to identify the rows associated with a single or set of observations from the **TAD**.

```
# {'y': array([[0.],
#             [0.],
#             [0.],
#             ...,
#             [0.],
#             [0.],
#             [0.]])},
# 'x': array([[ 1.,  0., 27., ...,  0.,  1.,  3.],
#            [ 2.,  0., 27., ...,  0.,  1.,  3.],
#            [ 3.,  0., 27., ...,  0.,  1.,  3.],
#            ...,
#            [11.,  1., 24., ...,  0.,  1.,  1.],
#            [12.,  1., 24., ...,  0.,  1.,  1.],
#            [13.,  1., 24., ...,  0.,  1.,  1.]])},
# 'w': array([[1.],
#            [1.],
#            [1.],
#            ...,
#            [1.],
#            [1.],
#            [1.]])},
# 'coord': array([ 0,  0,  0, ..., 431, 431, 431])}
```

The next step is to initialize the model, which involves setting several parameter values. The key considerations when initializing the models is number of trees and the split rules. The number of trees controls how many regression trees will be used. Typically 50 trees is a good default, but it can be adjusted to assist in model performance. Split rules is a specific PyMC-BART parameter and is used to designate the how the regression trees are constructed. The one requirement of the split rules is that the time covariate has to be set as a `pmb.ContinuousSplitRule()`. Otherwise, generally continuous variables can assigned `pmb.ContinuousSplitRule()` and categorical variables assigned `pmb.OneHotSplitRule()`. It is recommended to review the PyMC-BART literature for more information regarding parameterization of the models.

```
#####
# Instantiate the BART models
# model_dict defines specific model parameters
model_dict = {"trees": 50,
              "split_rules": [
                  "pmb.ContinuousSplitRule()", # time
                  "pmb.OneHotSplitRule()", # fin
                  "pmb.ContinuousSplitRule()", # age
                  "pmb.OneHotSplitRule()", # race
                  "pmb.OneHotSplitRule()", # wexp
                  "pmb.OneHotSplitRule()", # mar
                  "pmb.OneHotSplitRule()", # paro
                  "pmb.ContinuousSplitRule()", # prio
              ]
}
# sampler_dict defines specific sampling parameters
sampler_dict = {
    "draws": 200,
    "tune": 200,
```

```

        "cores": 8,
        "chains": 8,
        "compute_convergence_checks": False
    }
    BSM = sb.BartSurvModel(
        model_config=model_dict,
        sampler_config=sampler_dict
    )
304 The model can then be trained with the TAD input and predicted  $p_{ij}$  values yielded with the
305 PAD input.

#####
# Fit Model with TAD
BSM.fit(
    y = trn["y"],
    X = trn["x"],
    weights=trn["w"],
    coords = trn["coord"],
    random_seed=5
)

# Get posterior predictive for evaluation using the PAD
post1 = BSM.sample_posterior_predictive(
    X_pred=post_test["post_x"],
    coords=post_test["coords"]
)

306 Finally the survival probability can be derived from the  $p_{ij}$  estimates.

# Convert to SV probability.
sv_prob = sb.get_sv_prob(post1)
print(sv_prob["sv"].shape)
# (1600, 432, 13)

307 The sv_prob object above is a dictionary containing numpy arrays of both the  $p_{ij}$  and  $s_{ij}$ 
308 estimates, labeled "prob" and "sv" respectively. The  $p, s$  arrays are three dimensional with the
309 dimensions of the arrays being: - axis 0 = draws of the posterior predictive distribution: 1600 -
310 axis 1 = observations  $i$ : 432 - axis 2 = times  $j$ : 13

311 These arrays can be easily reduced down to point estimates and credible intervals using basic
312 numpy methods. For example to get the estimate of the mean over all observations, first get
313 the mean over the observations (axis 1) followed by the mean over the posterior draws (axis
314 0). The results being the estimated mean survival over the 13 time intervals.

315 Similarly the 0.05-0.95 credible interval for the estimated mean survival can be returned as the
316 quantile evaluations of the same mean-over-axis-1 array. This yields a (2,13) array with the
317 lower and upper bounds (rows) of the credible interval defined for each time point (columns).

# get the mean value across observations for each time within each draw of the posterior
ave_obs = sv_prob["sv"].mean(axis=1)
print(ave_obs.shape)
# (1600, 13)

# get the average across the posterior draws
ave_obs_draws = ave_obs.mean(0)
print(ave_obs_draws)
#[0.98282867 0.96457594 0.94578149
# 0.92613075 0.90503023 0.88429523

```

```
# 0.86382304 0.84376736 0.82281631
# 0.80169965 0.78080675 0.75948555
# 0.73816541]

# get the .05 and .95 percentiles of the mean across posterior draws
ci_obs_draws = np.quantile(ave_obs, [0.05, 0.95], axis=0)
print(ci_obs_draws)
# lower bound
# [0.97813492 0.95597036 0.93449701
# 0.91273325 0.88996273 0.86803963
# 0.84529644 0.82373944 0.8013158
# 0.77941446 0.75844137 0.73615453
# 0.71387    ]
# upper bound
# [0.9879464 0.97269846 0.95684153
# 0.93988969 0.91999613 0.90054844
# 0.8815908 0.86294746 0.84339772
# 0.82276706 0.80264632 0.78084766
# 0.76145384]
```

318 Examples of generation of marginal effect estimates can be found in the example notebooks  
319 provided in the repository documentation.

## 320 Conclusion

321 BART-Survival provides the computational methods required for completing non-parametric  
322 discrete-time survival analysis. This approach can have several advantages over alternative  
323 survival methods. These advantages include capabilities to incorporate non-linear and interac-  
324 tion effects into the model, naturally ability to regularize the model (which reduces the risk of  
325 over-fitting) and of being robust to issues of multi-collinearity. The BART-Survival approach  
326 is especially useful when the assumptions of alternative survival methods are violated.

327 Our BART-Survival algorithm has been tested in a rigorous simulation study, with additional  
328 applications to real-world data. While the manuscript for this work is currently under develop-  
329 ment, the results indicate similar performance as the the R-based BART survival method across  
330 settings of varied complexity. Both methods demonstrate the previously describe advantages  
331 over other survival approaches (such as Cox Proportional Hazard Models) when the relationships  
332 within the data becomes more complex or assumptions of the these other models are violated.  
333 A comparison of the R-based method and our BART-Survival algorithm is included in the  
334 [examples folder of the github repository](#).

335 Our library provides a convenient API for completing discrete-time survival analysis, along with  
336 the functionality to customize the methodology as needed. The associated API documentation  
337 can be found [here](#), along with the associated github repository [BART-Survival](#).

## 338 Acknowledgements

339 We thank Oscar Rincón-Guevara for helpful suggestions and review. We also thank Tegan  
340 Boehmer, Sachin Agnihotri, and Matt Ritchey for supporting the project throughout its  
341 development.

## References

- 342
- 343 Abril-Pla, O., Andreani, V., Carroll, C., Dong, L., Fonnesbeck, C. J., Kochurov, M., Kumar,  
344 R., Lao, J., Luhmann, C. C., Martin, O. A., Osthege, M., Vieira, R., Wiecki, T., & Zinkov,  
345 R. (2023). PyMC: A modern, and comprehensive probabilistic programming framework in  
346 Python. *PeerJ Computer Science*, 9, e1516. <https://doi.org/10.7717/peerj-cs.1516>
- 347 Altman, D. G., & Bland, J. M. (1998). Statistics Notes: Time to event (survival) data. *BMJ*,  
348 317(7156), 468–469. <https://doi.org/10.1136/bmj.317.7156.468>
- 349 Bradburn, M. J., Clark, T. G., Love, S. B., & Altman, D. G. (2003). Survival Analysis Part II:  
350 Multivariate data analysis – an introduction to concepts and methods. *British Journal of*  
351 *Cancer*, 89(3), 431–436. <https://doi.org/10.1038/sj.bjc.6601119>
- 352 Chipman, H. A., George, E. I., & McCulloch, R. E. (2010). BART: Bayesian additive regression  
353 trees. *The Annals of Applied Statistics*, 4(1). <https://doi.org/10.1214/09-AOAS285>
- 354 Cox, D. R. (1972). Regression Models and Life-Tables. *Journal of the Royal Statistical Society*  
355 *Series B: Statistical Methodology*, 34(2), 187–202. [https://doi.org/10.1111/j.2517-6161.](https://doi.org/10.1111/j.2517-6161.1972.tb00899.x)  
356 [1972.tb00899.x](https://doi.org/10.1111/j.2517-6161.1972.tb00899.x)
- 357 Harrell, F. E. (2015). *Regression Modeling Strategies: With Applications to Linear Models,*  
358 *Logistic and Ordinal Regression, and Survival Analysis*. Springer International Publishing.  
359 <https://doi.org/10.1007/978-3-319-19425-7>
- 360 Ishwaran, H., Kogalur, U. B., Blackstone, E. H., & Lauer, M. S. (2008). Random survival  
361 forests. *The Annals of Applied Statistics*, 2(3). <https://doi.org/10.1214/08-AOAS169>
- 362 Joffe, E., Coombes, K. R., Qiu, Y. H., Yoo, S. Y., Zhang, N., Bernstam, E. V., & Kornblau, S.  
363 M. (2013). Survival Prediction In High Dimensional Datasets – Comparative Evaluation  
364 Of Lasso Regularization and Random Survival Forests. *Blood*, 122(21), 1728–1728.  
365 <https://doi.org/10.1182/blood.V122.21.1728.1728>
- 366 Quiroga, M., Garay, P. G., Alonso, J. M., Loyola, J. M., & Martin, O. A. (2023). *Bayesian*  
367 *additive regression trees for probabilistic programming* (No. arXiv:2206.03619). arXiv.  
368 <https://doi.org/10.48550/arXiv.2206.03619>
- 369 Rossi, R. A. B., P. H., & Lenihan, K. J. (1980). Money, work, and crime: Some experimental  
370 results. *New York: Academic Press*.
- 371 Sparapani, R. A., Logan, B. R., McCulloch, R. E., & Laud, P. W. (2016). Nonparametric  
372 survival analysis using Bayesian Additive Regression Trees (BART). *Statistics in Medicine*,  
373 35(16), 2741–2753. <https://doi.org/10.1002/sim.6893>
- 374 Sparapani, R., Spanbauer, C., & McCulloch, R. (2021). Nonparametric Machine Learning and  
375 Efficient Computation with Bayesian Additive Regression Trees: The BART R Package.  
376 *Journal of Statistical Software*, 97(1). <https://doi.org/10.18637/jss.v097.i01>
- 377 Stel Vianda S., T. G., Dekker Friedo W. (2011). Survival analysis i: The kaplan-meier method.  
378 *Nephron Clin Pract*, 119(1). <https://doi.org/10.1159/000324758>