# INTERPRETABLE MACHINE LEARNING WITH APPLICATIONS IN HEALTHCARE

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF MANAGEMENT SCIENCE AND ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Mike Van Ness
June 2025

# Acknowledgments

First, I would like to thank my advisor Madeleine Udell. I started my PhD in the Statistics department at Cornell, and Madeleine and I started working together around the end of my first year. Little did I know that about a year later she would be moving to Stanford and that I would have to opportunity to move with her. Madeleine, thank you so much for always believing in my ability as a researcher, even when I did not believe in my own abilities. Unlike seemingly most PhD students these days, I did almost no research at all before my PhD, and yet you welcomed me with open arms into your lab. I'm not sure what you saw in me at the time, but I'm extremely thankful to have gotten that opportunity. For the first few years, I really struggled a lot with believing that I was capable of doing PhD level research. Yet, you always pushed me to continue forward, and eventually I found my footing and we started getting papers accepted. I think we've made for a really good team, and I'm really happy to have completed with journey under your guidance.

Next, I would like to thank my remaining committee members: Alex Sandhu, Trevor Hastie, Lu Tian, and Itai Ishlagi.

Alex, thank you for being a great collaborator throughout my PhD. During my time at Cornell, I was really hoping to start working on more applied projects, but couldn't find anyone interested in working together. Thus, I was really blessed to stumble into this opportunity to work with you on heart failure prediction pretty soon after I arrived at Stanford. You continue to amaze me with how much you know about statistics, despite being an expert primarily in cardiology. Thank you for spending time over the year helping me learn about do healthcare research.

Trevor, thank you greatly for taking the time to serve on my reading committee. Your work on additive models laid the foundation for most of the methodological work that I ended up doing in my PhD, and so I'm very grateful to have you be a part of my PhD.

Lu, thank you for helping me understand more about survival analysis in the early days of my journey into the topic. Your insights were very valuable for my understand and subsequent research. I've very thankful to have you as part of my committee.

Itai, thank you so much for your flexibility and willingness to join the committee so late. I really appreciate your time, and am so glad to have you as part of the committee.

Throughout this journey, I'm very thankful to have spent time with so many quality people who I still consider to be friends. First, my friends Kim Webb, Quinn Simonis, Steve Broll, Georgia Smits, and Derek

Hoare from my cohort at Cornell. I'm very thankful to have met you all and spent amazing quality time together before my move to Stanford. You all helped me get through some difficult classes at Cornell, and I'm very grateful for that. At Stanford, I've met many great friends that I've had the great pleasure of spending lots of time with since my move here. To give an incomplete list: Anders Wikum, Josh Grossman, Noah Benjamin-Pollak, my MS&E 125 CAs Alex Patel, Louisa Edwards, Cami Nicollier Sanchez, and Elizabeth Griffin, Zach Frangella, Pratik Rathore, Eva Zhang, Oscar Aguilar, Keertana Chidambaram, Mohak Goyal, Sahasrajit Sarmasarkar.

Perhaps most importantly, though, I must thank my family. First, my parents Steve Van Ness and Jaime Polk, who have given me so much over the years. Mom and Dad, thank you so much for making this academic journey possible. Your committee to my education, both financially and emotionally, unlocked so many doors in my life that would simple not be possible otherwise. I'm forever grateful for everything you have done for me over the years, and I will never be able to repay you for it.

And lastly, my amazing partner Trisha Agarwal. Somehow we had the amazing fortune of staying together throughout this whole process despite my mid-PhD move, and I am so very grateful for that. You've been my number 1 supporter the entire time, and also provided me so much emotional support over the years. I don't know how I would have been able to get through this PhD without you. Thank you so much for everything you do for me.

# Contents

# List of Tables

# List of Figures

# Introduction

The healthcare domain presents unique challenges for data-driven decision making. Electronic health records (EHRs) are characterized by heterogeneous data types, high dimensionality with hundreds to thousands of potential features, and significant amounts of missing values. Moreover, healthcare applications demand models that are not only accurate but also transparent and interpretable to support clinical decision-making and maintain trust among healthcare providers and patients. Traditional black-box machine learning approaches often fall short in providing the explanations necessary for clinical adoption. This dissertation aims to bridge the gap between the predictive power of modern machine learning techniques and the interpretability requirements of healthcare applications.

Chapter 1 presents a comprehensive investigation of missing value handling in prediction models, with a particular focus on the Missing Indicator Method (MIM) and a novel extension called Selective MIM (SMIM). Missing values are ubiquitous in healthcare data, and this chapter provides both theoretical guarantees and empirical evidence that MIM can effectively leverage the signal in missing patterns while maintaining model interpretability.

Chapter 2 introduces multiple novel approaches for interpretable survival analysis, a critical task in healthcare for predicting time-to-event outcomes such as mortality, disease recurrence, or hospital readmission. We present three glass-box survival models that achieve competitive predictive performance while maintaining full interpretability. The third model, called DNAMite, is a piece-wise constant neural additive model that is interpretable, accurate, and calibrated. DNAMite also natively incorporates MIM to handle missing values but without requiring imputation or additional features.

Chapter 3 describes the dnamite Python package, which implements the DNAMite model for regression, classification, and survival analysis tasks. This package provides researchers and practitioners with accessible tools to apply the methodologies developed in this dissertation to real-world healthcare problems.

Throughout this work, we demonstrate that interpretable machine learning models can achieve performance comparable to black-box counterparts while providing crucial transparency for healthcare applications. The methodologies presented here not only advance the state-of-the-art in interpretable machine learning but also offer practical solutions for analyzing healthcare data with missing values and building interpretable predictive models for time-to-event outcomes.

# Chapter 1

# Handling Missing Values in Prediction Models

## 1.1 Introduction

Missing data is an unavoidable consequence of tabular data collection in many domains. Nonetheless, most statistical studies and machine learning algorithms not only assume complete data, but cannot run on data sets with missing entries. Two common preprocessing methods are used to address this issue. The first method is complete case analysis, in which the fully observed data samples are used, while the others are discarded. However, discarding data reduces statistical power and can bias results, as partially-observed data samples may still contain important information. Furthermore, sometimes *all* data samples have missing values, in which case discarding incomplete samples is catastrophic.

The second method is missing value imputation, in which missing values are replaced by estimates from the observed data. While missing value imputation is well studied and widely used [107, 66], it comes with two problems. Firstly, most imputation methods are only (provably) effective when missing values are missing at random (MAR), which is often violated in real-world data, e.g. medical data [79]. In particular, many real-world data sets exhibit *informative missingness*, i.e. when the pattern of missing data encodes information about the response variable. When data has informative missingness, the MAR assumption is not typically satisfied, making imputation methods less effective.

Secondly, imputation methods are typically evaluated by their reconstruction error. This method of evaluation is reasonable when the task of interest itself is imputation, e.g. for recommender systems. However, optimal imputation in terms of lowest reconstruction error is not necessarily required for optimizing downstream prediction accuracy. In fact, recent theoretical work has shown that simple imputation schemes can still result in Bayes optimal predictions if the imputation function is paired with an appropriate prediction function [49, 60, 10]. Thus, it is possible to obtain optimal predictions without accurate imputations, which is particularly useful when making accurate imputations is challenging, i.e. for MNAR data.

An alternative strategy, particularly when missing values are informative, is the Missing Indicator Method (MIM), which directly leverages the signal in the missingness itself rather than optimizing imputation accuracy. For each partially observed feature, MIM adds a corresponding indicator feature to indicate whether the feature is missing or not. MIM can be used with any imputation method, but is most commonly used with mean imputation, which is equivalent to 0-imputation after centering features. MIM is a common method in practice, e.g. it is implemented in scikit-learn [78], yet MIM remains understudied from an empirical and, in particular, theoretical perspective.

In this chapter, we show both theoretically and empirically that MIM is an effective strategy for supervised learning in a wide variety of data regimes. While some previous work has studied MIM related to statistical inference [53, 33], our work focuses on MIM as part of a supervised learning pipeline, where prediction accuracy is the primary objective opposed to inference. Additionally, to better handle high-dimensional data sets where extra uninformative indicator features can lead to overfitting, we introduce *Selective MIM* (SMIM), a novel MIM extension to adaptively select which indicators are useful to add to the data set.

## 1.2 Missing Values in Supervised Learning

In this section we introduce the problem statement and relevant work, and expand on how we build on existing work.

### 1.2.1 Notation

We use the following conventions. Upper case letters $X$ are random variables. Bold letters $\boldsymbol{X}$ are vectors, which by default are always column vectors. Subscripts $X_j$ denote components of $\boldsymbol{X}$, and superscripts $\boldsymbol{X}^{(i)}$ denote sample vectors in a data set. We use $n$ for the number of samples/rows, and $p$ for the number of features/columns. We reserve $i$ as an index on samples and $j$ as an index on features. $\boldsymbol{R}$ will denote indicator features corresponding to missing values in $\boldsymbol{X}$. The sets $\text{obs}(\boldsymbol{R}) = \{j : R_j = 0\}$ and $\text{miss}(\boldsymbol{R}) = \{j : R_j = 1\}$ indicate observed and missing components of a vector. $X_j \perp\!\!\!\perp X_k$ means $X_j$ and $X_k$ are independent as random variables.

### 1.2.2 Problem Statement

We consider the following supervised learning setup. Let $\boldsymbol{X} \in \mathbb{R}^p$ be a vector of $p$ features, and $Y \in \mathcal{Y}$ a response to be predicted from $\boldsymbol{X}$, where $\mathcal{Y} = \mathbb{R}$ for regression tasks and $\mathcal{Y} = \{0, 1, \ldots, k-1\}$ for k-class classification tasks. The vector $\boldsymbol{X}$ contains a complete set of values, in the sense that a value exists for each component of $\boldsymbol{X}$ even if that value is not observed. In reality, we observe $\boldsymbol{Z} \in \{\mathbb{R} \cup \{*\}\}^p$ where $Z_j = X_j$ when $X_j$ is observed and $Z_j = ` * '$ when $X_j$ is unobserved/missing. $\boldsymbol{Z}$ yields the random binary vector $\boldsymbol{R} \in \{0,1\}^p$ that indicates the missing and observed components of $\boldsymbol{Z}$. Specifically, $R_j = 0$ when $X_j$ is observed and $R_j = 1$ when $X_j$ is missing.

Given a loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ and a distribution over pairs $(\boldsymbol{Z}, Y) \sim \mathcal{D}$, the goal is to find function

$f : \{\mathbb{R} \cup \{*\}\}^p \to \mathcal{Y}$ that minimizes the expected loss:

$$f^* = \underset{f}{\operatorname{argmin}} \; \mathbb{E}_{\mathcal{D}} \left[ \mathcal{L}(f(\boldsymbol{Z}), Y) \right]. \tag{1.1}$$

We will often drop the subscript $\mathcal{D}$ from expectations with the default being that the expectation is with respect to $\mathcal{D}$. The function $f$ can be a pipeline that combines an imputation method with a prediction method, or can directly predict $Y$ from $\boldsymbol{Z}$ without imputation, e.g. fitting a different model per missing pattern.

### 1.2.3 Types of Missing Data

Traditionally, partially observed data is categorized by the missing mechanism, i.e. the distribution of $\boldsymbol{R}$ conditional on the complete data $\boldsymbol{X}$. The most typical categorization of missing mechanisms is [68]:

- **Missing Completely at Random (MCAR).** The missing pattern is random and independent of the data: $P(\boldsymbol{R} \mid \boldsymbol{X}) = P(\boldsymbol{R})$.
- **Missing at Random (MAR).** The missing pattern is independent of unobserved values $\boldsymbol{X}_{\text{miss}(\boldsymbol{R})}$ conditioned on observed values $\boldsymbol{X}_{\text{obs}(\boldsymbol{R})}$: $P(\boldsymbol{R} \mid \boldsymbol{X}) = P(\boldsymbol{R} \mid \boldsymbol{X}_{\text{obs}(\boldsymbol{R})})$.
- **Missing Not at Random (MNAR).** The missing pattern may depend on unobserved values.

The MCAR and MAR mechanisms are sometimes considered *ignorable* in the sense that these mechanisms may be ignored for likelihood-based inference [89]. Many imputation methods work well on MCAR or MAR data [108, 99, 41, 126], while MNAR data is more challenging for imputation methods and requires modeling the missing mechanism itself [97, 72, 57].

While missing mechanisms define the relationship between $\boldsymbol{R}$ and $\boldsymbol{X}$, informative missingness is defined by the relationship between $\boldsymbol{R}$ and $Y$:
**Definition 1.2.1.** *For $(\boldsymbol{Z}, Y) \sim \mathcal{D}$ with missing pattern $\boldsymbol{R}$, the missing pattern is* informative *if $\boldsymbol{R} \not\perp\!\!\!\perp Y$.*

Assuming $\boldsymbol{X} \not\perp\!\!\!\perp Y$, this definition implies that MAR and MNAR data always have informative missingness. MCAR data can also have informative missingness if $\boldsymbol{R} \not\perp\!\!\!\perp Y$ despite $\boldsymbol{R} \perp\!\!\!\perp \boldsymbol{X}$, although this is rare in practice. In previous literature, informative missingness is most often associated with MNAR data, as in this setting the missingness is most informative [90, 64]. In general, we expect MIM to improve predictions when missingness is informative. Yet, we can have MAR data and not see any benefit to MIM over imputation alone if $\boldsymbol{R} \perp\!\!\!\perp Y \mid \boldsymbol{X}_{obs}$

### 1.2.4 Related Work

Missing values have been an important topic of study in statistics for decades. Classical work focused on statistical inference problems [89, 23, 68], while more recent work has focused more on imputation, ranging from statistical methods [41, 125] to iterative approaches [108, 99] and deep learning [124, 72, 30].

Even though MIM is commonly used in practice (e.g. it is implemented in sklearn [78]), its properties are surprisingly understudied in statistical and machine learning literature, particularly from the perspective of supervised learning. MIM has been explored more in the medical literature due to the ubiquity of missing values in medical data sets, and particularly the tendency of these missing values to be informative. A few medical papers advocate for MIM with mean imputation [94, 96], and some for MIM with other imputation methods [95, 84]. Other medical papers caution the use of MIM when statistical estimation and inference is the downstream task, as then using MIM can add bias to parameter estimates [53, 33]. We instead focus on the problem of optimal supervised learning.

Many previous methodological and empirical works have studied missing value preprocessing for supervised learning, yet most do not include MIM or a similar method to capture informative missingness. Some empirical studies have evaluated combinations of imputation and prediction models [26, 120], but these most often exclude MIM. Recent AutoML methods attempt to optimize the entire supervised learning pipeline [122, 27, 25], but without considering MIM. Deep learning can be used to jointly optimize imputation and prediction if both models are neural networks [59, 46], but these methods do not capture informative missingness without additionally using missing indicators. Decision trees can uniquely handle missing values without imputation, e.g. using the Missing Incorporated as Attribute method [104, 49], but such methods are only applicable to tree-based methods.

Lastly, some recent papers have explored impute-then-predict pipelines from a theoretical perspective. [49, 60, 10] show that accurate imputation is not needed to produce Bayes optimal predictions as long as a powerful enough predictor is used. While this theory is not immediately applicable in practice, since constructing these predictors would often be impossible, it does give motivation for developing missing value preprocessing methods that do not focus on imputation accuracy, such as MIM with mean imputation. Perhaps the most relevant previous work to this paper is [61], which gives a risk bounds for linear models with 0-imputation using both MIM and an expanded linear model. We also present theory for MIM with linear models, but under different assumptions and focusing on asymptotics rather than risk bounds. We also perform a much more diverse set of empirical experiments compared to [61].

## 1.3 Missing Indicator Method Theory

In this section, we present a theoretical study of MIM for linear regression. Following the notation from Section 1.2, let $\boldsymbol{X} \in \mathbb{R}^p$ be $p$ features, $Y \in \mathbb{R}$ a continuous response, and $\boldsymbol{Z} \in \{\mathbb{R} \cup \{*\}\}^p$ the observed features with potentially missing values. The full optimization problem in Equation 1.1 with squared error loss $\mathcal{L}(x, y) = (x-y)^2$ is solved by the conditional expectation $\mathbb{E}[Y \mid \boldsymbol{Z}]$. However, as explained in [61, 59], this conditional expectation is combinatorial in nature, as it involves estimating a different expectation for each missing pattern $\boldsymbol{R}$:

$$f^* = \mathbb{E}[Y \mid \boldsymbol{Z}] = \mathbb{E}[Y \mid \boldsymbol{X}_{\mathrm{obs}(\boldsymbol{R})}, \boldsymbol{R}] \tag{1.2}$$

$$= \sum_{\boldsymbol{r} \in \{0,1\}^d} \mathbb{E}[Y \mid \boldsymbol{X}_{\mathrm{obs}(\boldsymbol{r})}, \boldsymbol{R} = \boldsymbol{r}] \mathbb{1}_{\boldsymbol{R}=\boldsymbol{r}}. \tag{1.3}$$

In [61, 59], the conditional expectation in Equation 1.2 is estimated after assuming that $\boldsymbol{X}$ comes from a Gaussian distribution. In this chapter, we make no distributional assumptions, and instead make the following assumptions:

**Assumption 1.3.1.** *$Y$ is centered as $Y \leftarrow Y - \mathbb{E}[Y]$, while $\boldsymbol{Z}$ is centered over the observed entries $Z_j \leftarrow Z_j - \mathbb{E}[Z_j \mid R_j = 0]$ and imputed with 0, resulting in the imputed data vector $\tilde{\boldsymbol{Z}}$ defined as*

$$\tilde{Z}_j = \begin{cases} X_j & R_j = 0, \\ 0 & R_j = 1. \end{cases} \tag{1.4}$$

**Assumption 1.3.2.** *$f$ is constrained to be a linear function, and thus $f^*$ is the best linear prediction function.*

We now examine $f^*$ with and without MIM. Specifically, we compare the best linear predictors (BLPs):

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, \mathbb{E}[(Y - \tilde{\boldsymbol{Z}}^T\boldsymbol{\beta})^2], \tag{1.5}$$

$$\boldsymbol{\beta}^*_{MIM},\ \boldsymbol{\gamma}^*_{MIM} = \underset{(\boldsymbol{\beta},\boldsymbol{\gamma})}{\operatorname{argmin}}\, \mathbb{E}[(Y - \tilde{\boldsymbol{Z}}^T\boldsymbol{\beta} - \boldsymbol{R}^T\boldsymbol{\gamma})^2]. \tag{1.6}$$

In practice, we would estimate these BLPs using a training set $\{(\boldsymbol{Z}^{(1)}, \boldsymbol{R}^{(1)}), \ldots (\boldsymbol{Z}^{(n)}, \boldsymbol{R}^{(n)})\}$, obtaining the standard ordinary least squares (OLS) estimates $\hat{\boldsymbol{\beta}}$, $\hat{\boldsymbol{\beta}}_{MIM}$, and $\hat{\boldsymbol{\gamma}}_{MIM}$. Under mild conditions, most notably that $\mathbb{E}[\tilde{\boldsymbol{Z}}\tilde{\boldsymbol{Z}}^T]$ and $\mathbb{E}[(\tilde{\boldsymbol{Z}}^T, \boldsymbol{R}^T)^T(\tilde{\boldsymbol{Z}}^T, \boldsymbol{R}^T)]$ are invertible, these OLS estimates will converge to the corresponding BLPs in Eqs. (1.5) and (1.6); see [34] for further details. This convergence validates the study of the best linear predictors, as they serve as the limits of the OLS estimates obtained in practice.

Theorem 1.3.1 starts by analyzing the simple case when $p = 1$, which has particularly nice properties.

**Theorem 1.3.1.** *Grant Assumptions 1.3.1 and 1.3.2. Let $p = 1$, and let $\mathcal{M} = \{i\colon R^{(i)} = 1\}$ be the missing samples, then the OLS estimates are*

$$\hat{\beta}_{MIM} = \hat{\beta}, \quad \hat{\gamma}_{MIM} = \frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}} Y^{(i)}, \tag{1.7}$$

*and thus the BLPs are*

$$\beta^*_{MIM} = \beta^*, \quad \gamma^*_{MIM} = \mathbb{E}[Y \mid R = 1]. \tag{1.8}$$

A direct corollary of Theorem 1.3.1 is that if $R$ is uninformative, then

$$\gamma^*_{MIM} = \mathbb{E}[Y \mid R = 1] = \mathbb{E}[Y] = 0 \tag{1.9}$$

since $Y$ is centered. On the other hand, if $R$ is informative, then $\gamma^*_{MIM} = \mathbb{E}[Y \mid R = 1] \neq 0$, allowing the model to adjust its predictions when $X$ is missing. Specifically, because $X$ is imputed with 0, the best linear prediction function is

$$f^*_{linear}(X) = \begin{cases} X\beta^* & \text{if } R = 0, \\ \mathbb{E}[Y \mid R = 1] & \text{if } R = 1. \end{cases} \tag{1.10}$$

or correspondingly in finite sample

$$\hat{f}_{linear}(X) = \begin{cases} X\hat{\beta} & \text{if } R = 0, \\ \frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}} Y^{(i)} & \text{if } R = 1. \end{cases} \tag{1.11}$$

We see that when $X$ is observed, the model ignores $R$ as a feature, and when $X$ is missing, the model predicts the average of $Y$ among the missing values. Note that when $p = 1$, this result occurs both in finite sample and asymptotically.

We now consider the more general case of $p > 1$ features in Theorem 1.3.2.

**Theorem 1.3.2.** *Grant Assumptions 1.3.1 and 1.3.2.*

*(a) If the missing mechanism is MCAR and $\boldsymbol{R}$ is uninformative, then*

$$\boldsymbol{\beta}^* = \boldsymbol{\beta}^*_{MIM}, \quad \boldsymbol{\gamma}^*_{MIM} = 0. \tag{1.12}$$

*(b) If (i) the missing mechanism is self-masking, i.e. $P(\boldsymbol{R} \mid \boldsymbol{X}) = \prod_j P(R_j \mid X_j)$; (ii) $X_j \perp\!\!\!\perp X_k$ for $j \neq k$; and (iii) $\boldsymbol{R}$ is centered, then*

$$\boldsymbol{\beta}^* = \boldsymbol{\beta}^*_{MIM}, \tag{1.13}$$

$$\tag{1.14}$$

*and for $j = 1, \ldots, p$*

$$\gamma^*_{MIM_j} = \mathbb{E}[Y \mid X_j \textit{ missing}] - \mathbb{E}[Y \mid X_j \textit{ observed}]. \tag{1.15}$$

*(c) If $\{1, \ldots, p\}$ can be partitioned into $d$ blocks $B_1, \ldots, B_d$ with block-independence: $R_j, X_j \perp\!\!\!\perp R_k, X_k$ when $j, k$ are in different blocks, then for $j \in B_m$*

$$\begin{aligned}\beta^*_{MIM_j} = \sum_{\ell \in B_m} a\mathbb{E}[\tilde{Z}_j Y] + \\ b(\mathbb{E}[Y \mid X_\ell \textit{ missing}] - \mathbb{E}[Y \mid X_\ell \textit{ observed}])\end{aligned} \tag{1.16}$$

*for $a$ and $b$ are functions of $(\tilde{\boldsymbol{Z}}, \boldsymbol{R})$, and the same for $\gamma^*_{MIM_j}$ but for different $a$ and $b$.*

The proofs of Theorems 1.3.1 and 1.3.2 are given in Section 1.9. Theorem 1.3.2 (a) shows that when the missing mechanism is MCAR and $\boldsymbol{R}$ is uninformative, the best linear predictions are the same regardless of whether MIM is used or not. Since MIM is not expected to improve model performance when missingness is not informative, it is promising that MIM does not decrease the accuracy of linear models in this setting, at least asymptotically. This is important because practitioners are often unaware of the missing mechanism of the data, which is difficult to test [7]. It is therefore crucial to understand the effects of MIM across all missing mechanisms. This evidence that MIM does not hurt performance even in a worst-case scenario justifies its use when predictive performance is prioritized.

Parts (b) and (c) of Theorem 1.3.2 show that when missingness is informative, the best linear prediction function using MIM learns how the missing values impact the distribution of $Y$, thereby leveraging the signal in the missing values. In part (b), under the fully-independent self-masking mechanism, the form of $\boldsymbol{\gamma}^*_{MIM}$ in Equation 1.15 directly accounts for the informative missingness in each feature, i.e. $\mathbb{E}[Y \mid X_j \text{ missing}] - \mathbb{E}[Y \mid X_j \text{ observed}]$. Part (c) presents a more general case, where independence is only assumed blockwise. Nonetheless, the formula for the BLP in (c) still contains the same expression as in Equation 1.15

from (b), which encodes the (potential) informative missingness of each feature.

### 1.3.1 Related Theory

As discussed in Section 1.2.4, even though MIM is commonly used in practice, very little previous work has studied MIM theoretically, especially from the lens of supervised learning. The most similar work to ours is [61], which also presents theoretical results for linear regression with missing values. Much of the theory in [61] centers around understanding the Bayes predictor $f^*$ as in Eq.(1.2), assuming a linear generating function and Gaussian data. Further, Theorem 5.2 in [61] presents an upper bound for the risk of the linear model with MIM, i.e. the expected loss in Equation 1.6. [61] compares this risk bound to the risk bound for the "expanded" linear model, which fits a separate linear model for each missing pattern in Equation 1.3.

The theoretical results in [61] are complementary to the results in our work in multiple ways. First, our results make no assumptions about the distribution of the covariates as well as about the data generating model. Instead, we make different assumptions about the missing mechanism. Second, [61] focuses on understanding the form of the Bayes predictor and proving risk bounds, while our results specify the coefficient values for the best linear predictors. While the risk bounds in [61] are useful for comparing the finite-sample risk of MIM compared to other models, our results present a more practical understanding of linear model coefficients with MIM, and how these coefficients can directly encode or ignore missingness depending on the missing mechanism.

## 1.4 Selective MIM

Observe that MIM adds a missing indicator for *every* feature that is partially observed, regardless of the missing mechanism. Theorem 1.3.2 (a) justifies this behavior asymptotically, since the uninformative indicators can be ignored as $n \to \infty$. However, in finite sample, and particularly for high-dimensional data, adding many uninformative indicators can cause overfitting and thus can hurt model performance.

Keeping only the informative indicators from MIM would help stabilize MIM on high-dimensional data. One strategy to do such selection would be to use a statistical test to discover the missing mechanism, and use this information to determine which indicators to keep. However, testing for the missing mechanism is challenging. Previous literature has suggested tests for MCAR, notably Little's MCAR test [67], but these tests have been criticized [7]. Testing for MAR or MNAR, on the other hand, is impossible, since it would require access to the unobserved data. From Definition 1.2.1, however, we can instead test for informative missingness directly by testing the relationship between $\boldsymbol{R}$ and $Y$, and determine which indicators are worth including, instead of adding all.

We propose a novel strategy to selectively add indicators based on a statistical test, which we call *Selective MIM* (SMIM). The overall procedure for SMIM is summarized in Algorithm 1.1. Instead of testing the relationship between $R_j$ and $\boldsymbol{X}$, as a test for the missing mechanism would, SMIM tests the relationship between $R_j$ and $Y$, directly yielding signal from the informative missingness towards the response of each missing indicator. Since we assume $Y$ is complete, there are no issues testing the relationship between $R_j$ and $Y$, as there would be when testing for the missing mechanism. Further, if $X_j$ is correlated with $Y$, then

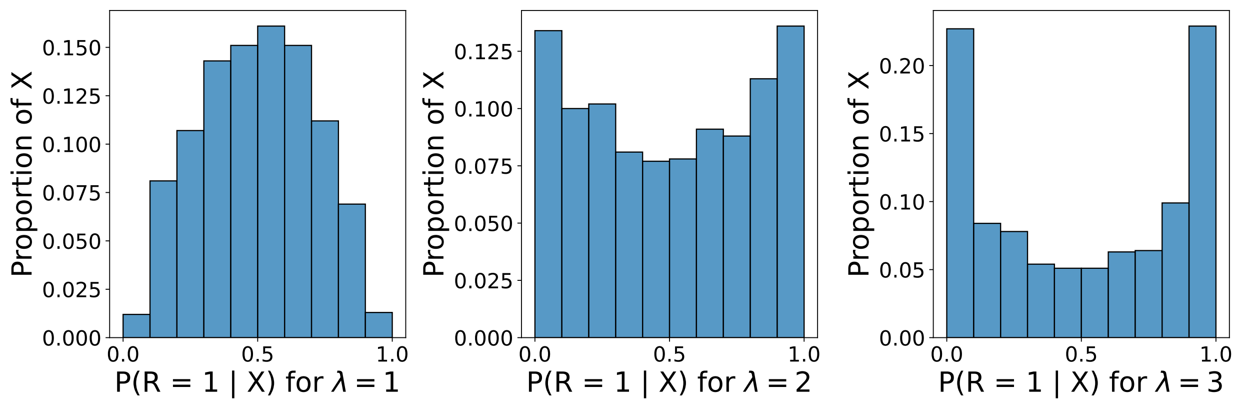


Figure 1.1: Distributions of masking probabilities $P(R = 1 \mid X)$ generated from $X \sim \mathcal{N}(0, 1)$ using Equation 1.17 for different values of the informativeness parameter $\lambda$. Larger values of $\lambda$ result in 'steeper' sigmoid functions in Equation 1.17, hence more values lie in the masked-with-high-probability regime.

testing the relationship between $R_j$ and $Y$ serves as a proxy for testing the relationship between $X_j$ and $R_j$, which is otherwise impossible if $X_j$ has missing values.

Specifically, for feature $j$, SMIM tests whether $R_j$ and $Y$ are independent. When $Y$ is continuous, we use a two-sample t-test comparing the $E[Y \mid R = 1]$ and $E[Y \mid R = 0]$. When $Y$ is discrete, we use a chi-squared test of independence between $Y$ and $R_j$. To correct for multiple testing across the features, we use Benjamini-Hochberg p-value correction [9] with false discovery rate (FDR) of $\alpha$. Since false negatives (not adding an indicator that is informative) are more harmful than false positives (adding an indicator that is not informative), we use a relatively high FDR of $\alpha = 0.1$ throughout the chapter. See Algorithm 1.1.

**Algorithm 1.1** Selective MIM

```
1: **Input:** Missing indicators $\boldsymbol{R}$, response $Y$, error rate $\alpha$.
2: **Output:** Indicator indices to keep $\mathcal{I} \subseteq \{1, \ldots, p\}$.
3: pvals ← [ ]
4: for $j = 1$ to $p$ do
5:     if $Y$ is continuous then
6:         pval ← t-test($Y \mid R_j = 0,\ Y \mid R_j = 1$)
7:     else if $Y$ is categorical then
8:         pval ← Chi2-test(Contingency-Table($Y, R_j$))
9:     end if
10:    pvals[$j$] ← pval
11: end for
12: reject = Benjamini–Hochberg(pvals, $\alpha$)
13: return $\mathcal{I} = \{j : \text{reject}_j = \text{true}\}$.
```

## 1.5 Experiments

We now empirically evaluate MIM and SMIM through experiments on synthetic data as well as real-world OpenML data sets with synthetic missing values. We show that MIM boosts performance when missing values are informative, and does not negatively affect performance on most data sets with uninformative missing values. Further, we show that SMIM can successfully discover the missing indicators that are informative,

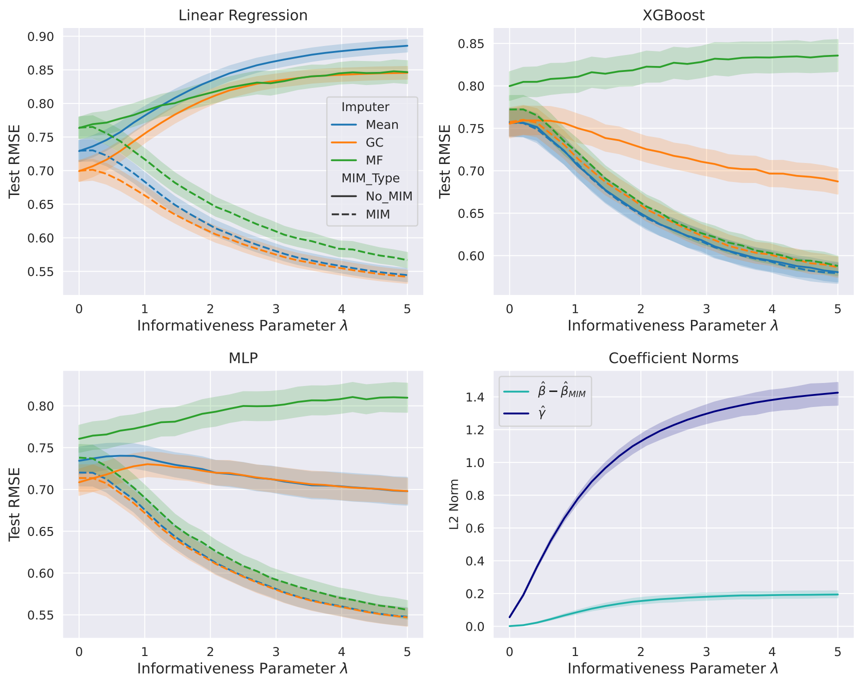


Figure 1.2: MIM performance on synthetic data with $(n, p) = (10000, 10)$ with different imputation methods, as a function of the informativeness parameter $\lambda$ (see Equation 1.17). The top left, top right, and bottom left plots demonstrate that MIM (dashed lines) reduces test RSME in almost all scenarios when missingness is informative. The bottom right plot shows the linear regression coefficient norms, where $||\hat{\boldsymbol{\gamma}}||$ increases as missing value become more informative, as shown in Theorem 1.3.2.

and is more effective than MIM on high-dimensional data sets.

### 1.5.1 Setup

For all experiments in this section, we start with complete data and mask values, controlling for how much the missing pattern is informative. To generate the missing masks, we use a self-masking mechanism:

$$\begin{aligned} P(R_j = 0 \mid \boldsymbol{X}) &= P(R_j = 0 \mid X_j) \\ &= \frac{1}{1 + \exp(-\lambda_j X_j)}. \end{aligned} \tag{1.17}$$

We call $\lambda_j$ the *informativeness parameter* as it directly controls how informative the missing values in $X_j$ are by determining the steepness of the sigmoid masking probability function. The higher the value of $\lambda_j$, the more informative the missing values are. If $\lambda_j = 0$ for all $j$, then the missing values are MCAR and the missingness is uninformative. The impact of $\lambda_j$ on $R_j$ is showcased in more detail in Figure 1.1.

We consider the following preprocessing methods to treat missing values:

- **Mean:** Mean imputation over the observed entries. This is equivalent to 0-imputation, since we always standardize features.
- **MF:** Imputation via missForest [99], an iterative imputer based on random forests [12].
- **GC:** Imputation via gcimpute [126, 127], an EM-style imputer which uses the Gaussian Copula.

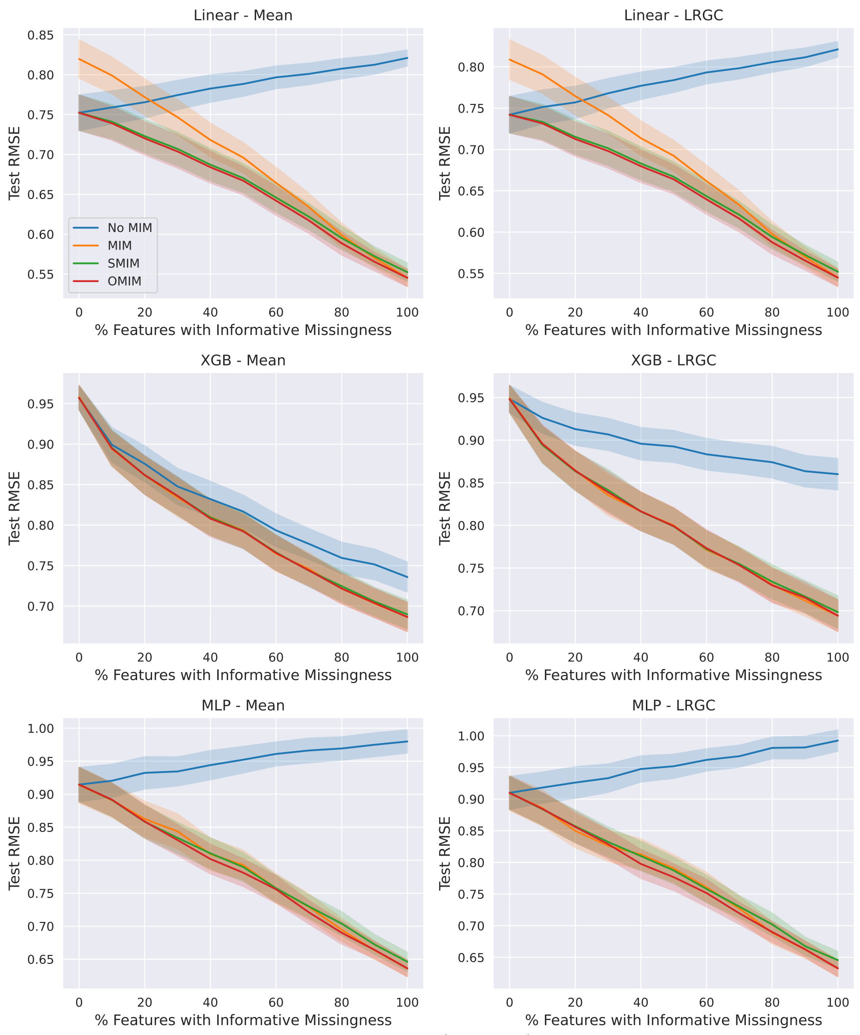


Figure 1.3: Comparison of MIM, SMIM, Oracle MIM (OMIM), and No MIM on high-dimensional synthetic data with $(n, p) = (10000, 1000)$, as a function of the percent of features with informative missingness.

- **LRGC:** Imputation via gcimpute with the low rank Gaussian Copula model [125], useful for high-dimensional data sets.
- **Imputer + MIM:** Imputation via the given imputer, along with MIM.
- **Imputer + SMIM:** Imputation via the given imputer, along with Selective MIM as in Algorithm 1.1.

We occasionally use the term "No MIM" to refer to imputation without MIM or SMIM. We choose these imputation methods to include either an iterative approach (MF) and an EM-based approach (GC), both popular classes of imputation methods. For supervised learning models, we use linear/logistic regression, XGBoost [18], and a multi-layer perceptron (MLP), giving us one model from the 3 most popular classes of supervised learning models (linear models, boosted decision trees, and neural networks). We standardize features over the observed entries in all experiments. We use a 75%-25% train-test data split and run each experiment for 20 trials, each with a different model seed, train-test split, and missing value mask. To isolate the impact of the missing value handling, we analyze how different missing value preprocessing methods impact each supervised learning method separately, rather than comparing the supervised learning methods

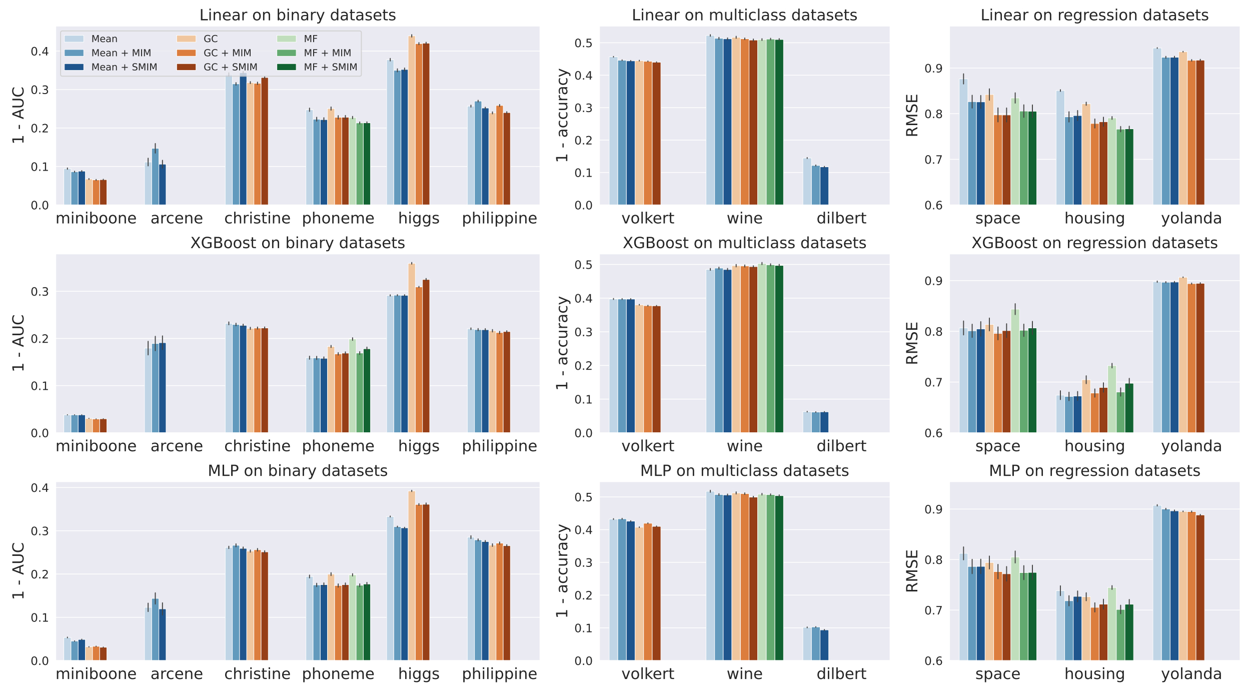


Figure 1.4: Test performance (lower is better) of MIM and SMIM with various imputation methods on OpenML data sets. Missing values are generated according to Equation 1.17, where for each feature $\lambda_j = 2$ with probability $p_{\text{inf}} = 0.5$ and $\lambda_j = 0$ otherwise. Performance metric is RMSE for regression problems, 1-AUC for binary classification problems, 1-accuracy for multi-class problems. Each bar represents the mean across 20 trials, and a a missing bar indicates a $> 3$ hour run time.

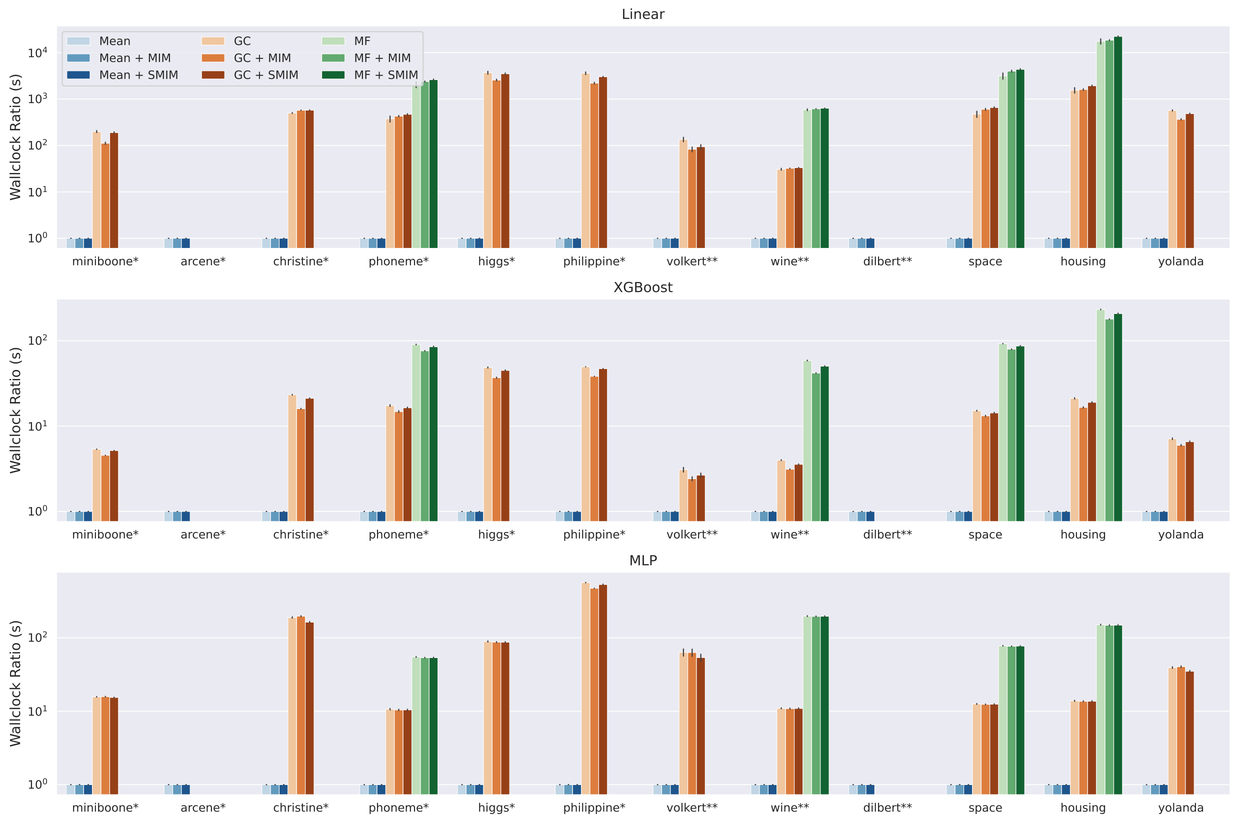


Figure 1.5: Wallclock times (in seconds) for results in Figure 1.4. Each time is the ratio between the indicated method and the corresponding mean imputation method. A missing bar indicates a $> 3$ hour run time, as in Figure 1.4.

to each other. For performance metrics, we use RMSE for regression tasks, 1 - AUC for binary classification tasks, and 1 - accuracy for multiclass classification tasks (so in all cases *lower is better*). We release the code necessary to reproduce our results [1].

### 1.5.2 MIM on Tree-Based Models

Before discussing our empirical results, we preface our findings with intuition for why XGBoost will likely behave differently with MIM than linear models and MLPs. Tree-based methods treat all features as discrete, and so they discretize continuous features using binning [13]. When using a constant imputation method like mean imputation, all imputed values always fall into the same bin and will always be split together by each tree. We thus expect MIM to provide little additional information to the tree-based methods in this setting, since splitting on the indicator feature can be alternatively achieved by splitting twice to isolate the bin with the constant imputation value. Meanwhile, if a non-constant imputation method is used, then MIM does add new splits for the tree to search over and thus has high potential to be valuable. On the other hand, linear and neural network models do not treat continuous features as discrete, and thus are expected to benefit from missing indicators with all imputation methods.

### 1.5.3 Synthetic Data

**Low Dimensional Data** We first consider the effects of MIM on synthetic data as a function of the informativeness parameter $\lambda$. Using $n = 10000$ and $p = 10$, we generate $\boldsymbol{X} \sim \mathcal{N}(0, \Sigma)$ with $\Sigma = \rho \mathbf{1}\mathbf{1}^T + (1-\rho)\mathcal{I}$ using $\rho = 0.3$ and $Y = \boldsymbol{X}^T \boldsymbol{\beta} + \epsilon$ for $\boldsymbol{\beta} \sim \mathcal{N}(0, \mathcal{I})$ and $\epsilon \sim \mathcal{N}(0, \sigma^2)$ with $\sigma^2$ chosen to enforce a signal-to-noise ratio of 10. We mask each feature according to Equation 1.17 with $\lambda_j = \lambda$ for all $j = 1, \ldots, p$. The results for $\lambda \in [0, 5]$ are shown in Figure 1.2. The top left, top right, and bottom left plots show that MIM continually reduces RSME as $\lambda$ increases across all imputation methods, which confirms that MIM is an effective preprocessor for informative missing values. The lone exception is XGBoost, which benefits from MIM using GC and MF imputation but not when using mean imputatation. This might be because mean imputation already allows trees to capture the informative signal in the missing values without MIM, as explained in Section 1.5.2.

The bottom right plot shows the linear regression coefficient norms $||\hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}_{MIM}||_2$ and $||\hat{\boldsymbol{\gamma}}||_2$ as a function of $\lambda$, using the definition of these coefficients from Section 1.3. When $\lambda = 0$, both norms are close to 0, which we expect from Theorem 1.3.2 (a) as it tells us that $\hat{\boldsymbol{\beta}} \approx \hat{\boldsymbol{\beta}}_{MIM}$ and $\hat{\boldsymbol{\gamma}} \approx 0$ when missingness is uninformative. As $\lambda$ increases, i.e. the missing values become more informative, $||\hat{\boldsymbol{\gamma}}||_2$ increases while $||\hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}_{MIM}||_2$ remains small. This behavior parallels Theorem 1.3.2 (b), which shows that under a self-masking mechanism like Eq. (1.17), $||\hat{\boldsymbol{\beta}} - \hat{\boldsymbol{\beta}}_{MIM}||_2$ should remain small but $||\hat{\boldsymbol{\gamma}}||_2$ should increase to capture the informative signal in each variable, i.e. $\mathbb{E}[Y \mid R_j = 1] - \mathbb{E}[Y \mid R_j = 0]$. It is noteworthy that this behavior still holds even without independence among the features in $\boldsymbol{X}$, suggesting that this behavior might hold, at least approximately, in more general linear regression settings.

**High Dimensional Data** We now consider high-dimensional synthetic data with $n = 10000$ and $p = 1000$. To generate realistic high-dimensional data, we generate $\boldsymbol{X} \sim \mathcal{N}(0, \Sigma)$ where $\Sigma$ is block diagonal with $b$ blocks

[1] `https://github.com/mvanness354/missing_indicator_method`.

and $d$ features per block, with block elements $\rho\mathbf{1}\mathbf{1}^T + (1-\rho)\mathcal{I}$ using $\rho = 0.5$. We compute $\boldsymbol{X}^* \in \mathbb{R}^b$ as the block-wise mean of $\boldsymbol{X}$ (averaging over features in each block), and generate $Y$ as in the low dimensional case: $Y = \boldsymbol{X}^{*T}\boldsymbol{\beta} + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma^2)$. This simulates real-world high-dimensional data where several features may be correlated with each other (but predominantly independent of other features), and thus are more likely to be missing together under informative missingness.

To generate the mask, we select only a random subset of blocks to have informative missingness: for each block, with probability $p_{\text{inf}}$ we set $\lambda = 2$ for all features in the block as a whole, and with probability $1 - p_{\text{inf}}$ we set $\lambda = 0$ for all features in said block. In this setting, SMIM should be able to detect which features have informative missing values and only add the correponding indicators. Along with MIM and SMIM, we also run experiments with Oracle MIM (OMIM), in which only features masked with $\lambda_j = 2$ are added, to represent an unrealistic but optimal preprocessor.

The results for $p_{\text{inf}} \in [0, 1]$ are shown in Figure 1.3. We use LRGC instead of GC since the data is high-dimensional, and we do not report MF because computations do not terminate in less than 3 hours. Using linear models, SMIM achieves comparable RMSE to OMIM and significantly better RMSE than No MIM across all values of $p_{\text{inf}}$. Further, SMIM outperforms MIM for smaller values of $p_{\text{inf}}$, when MIM adds many uninformative indicators. For MLP models, MIM and SMIM achieve comparable error to OMIM across all $p_{\text{inf}}$, demonstrating that MLPs can more readily ignore uninformative feature than linear models. Lastly, XGBoost with mean imputation is comparable for preprocessors, but suffers from No MIM with LRGC, further supporting the discussion on tree-based models in Section 1.5.2.

### 1.5.4 Masked Real-World Data

We now run experiments on fully-observed real-world data sets obtained from the OpenML data set repository [115]. We mask entries according to Equation 1.17, using $\lambda_j = 2$ with probability 0.5 and $\lambda_j = 0$ with probability 0.5 for all features $j$. We select 12 data sets that cover a diverse spectrum of values for $n$ (number of samples), $p$ (number of features), and outcome type (binary, multiclass, regression); see Section 1.10.1 for further OpenML data set descriptions. Lastly, we use LRGC for high-dimensional data sets (arcene, christine, philippine, volkert, dilbert, yolanda) and use GC for all other data sets.

Figure 1.4 shows the performance of each missing value preprocessing method paired with the 3 supervised learning models. For Linear and MLP models, MIM and SMIM improve performance across imputation methods for almost all data sets. This affirms the importance of MIM when missing values are informative. XGBoost generally does as well with mean imputation as with other imputation methods, and is less impacted by MIM, supporting the discussion from Section 1.5.2. On high-dimensional data sets, SMIM outperforms MIM in most cases. This further demonstrates the value of discarding uninformative features in high-dimensional data, which SMIM can do effectively.

To better understand the time efficiency of the methods employed, we plot the relative wallclock times of each result from Figure 1.4 in Figure 1.5. It is clear from Figure 1.5 that the choice of imputation method dominates the computation time, instead of whether or not MIM or SMIM is used. Specifically, mean imputation is orders of magnitude faster than GC and MF imputation. Also, MIM and SMIM appear to add relatively very little time to the total computation time, even though MIM doubles the number of features,

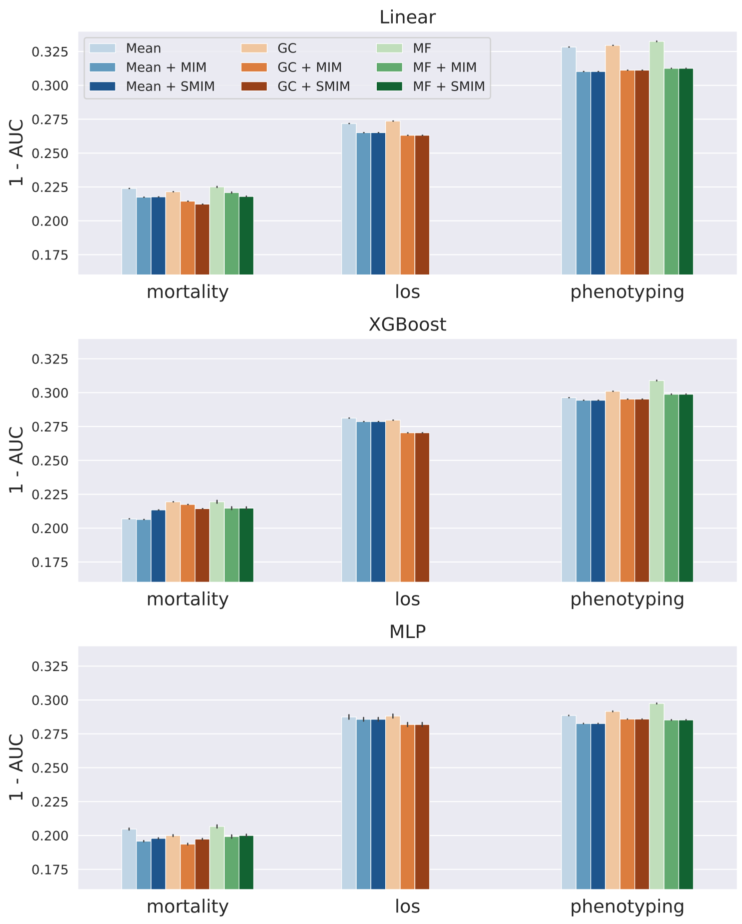


Figure 1.6: MIM and SMIM performance (lower is better) for clinical tasks on the MIMIC-III data set. Each bar represents the mean across 20 trials, each with a different random seed. A missing bar indicates a $> 3$ hour run time. MIM and SMIM improve performance for logistic regression and MLP models on all 3 tasks, following the behavior exhibited in Figure 1.4.

suggesting that the method of imputation is much more important than the inclusion of MIM. Further, notice that in Figure 1.4, (S)MIM with mean imputation often performs very comparably to (S)MIM with GC and MF. Therefore, (S)MIM with mean imputation is an effective yet remarkably efficient alternative to using expensive imputation models.

## 1.6 MIM on the MIMIC Benchmark

In Section 1.5, we demonstrate that MIM and SMIM are effective tools for capturing informative missingness in synthetic and real-world data sets with synthetic missing values. In this section, we demonstrate the effectiveness of MIM and SMIM on healthcare data which already has missing values. Specifically, we use MIMIC-III [48], an open source clinical database of electronic health records (EHRs) from the Beth Israel Deaconess Medical Center in Boston, Massachusetts. To preprocess the MIMIC data, we utilize the mimic3-benchmark [35], which generates tabular data for various clinical tasks. We consider the following tasks for our experiments:

- **In Hospital Mortality Prediction**: binary mortality prediction using the first 48 hours of an ICU stay.
- **Length-of-Stay Prediction (LOS)**: prediction of the remaining length-of-stay in the hospital. We formulate this as a binary prediction task to predict whether or not a patient will leave the hospital in the next 7 days.
- **Phenotyping**: prediction of the acute care conditions present during a given hospital stay, formulated as a multi-label binary classification problem for 25 phenotypes.

For each task, the target metric is 1 - AUC, following Section 1.5 and using a metric such that lower is better (for consistency across response types). For phenotyping, we average the AUC for each of the 25 labels to create the macro AUC score, then set the metric as 1 - macro AUC. The train and test splits are established by the benchmark and kept constant across all trials. The data set for each task contains 17 clinical variables that describe patient vital signs and other typical hospital measurements. For additional details about our these variables and well as our experimental setup for MIMIC experiments, see Section 1.10.2.

Figure 1.6 shows the results of MIM and SMIM with the prediction pipelines used in Section 1.5. As on synthetic data and OpenML data, MIM and SMIM both consistently improve predictive performance with linear and MLP models, as well as with XGBoost when using GC or MF for imputations. This result is particularly significant because it demonstrates that real-world data sets often have informative missing values, and MIM can help predictive models learn the signal in these missing values. Additionally, there are no scenarios MIM negatively impacts predictive performance, confirming that MIM should be a standard preprocessing tool for supervised learning on low-dimensional real-world data sets, should predictive performance be prioritized.

## 1.7 Discussion

From our experiments on synthetic data, real-world data with synthetic missing values, and real-world EHR data, we obtain the following main takeaways:

- When missing values are informative, MIM increases predictive performance of linear models and neural networks. This is supported by Figure 1.2, where increasing the informativeness in synthetic data gradually increases the effectiveness of MIM; in OpenML data with informative missing values in Figure 1.4; and in EHR data in Figure 1.6.
- Tree-based models in general benefit less from MIM than linear models and neural networks, particularly when using mean imputation. We provide a possible explanation for this behavior in Section 1.5.2, and see this behavior manifest across all of our experiments.
- The only scenario where MIM might harm predictive performance is on high-dimensional data sets, e.g. the synthetic data in Figure 1.3 and on select high-dimensional OpenML data sets in Figure 1.4. In these cases, Selective MIM (SMIM) is a stable extension of MIM that adds all the indicator features that have informative missing values.
- MIM and SMIM can increase performance not only with mean imputation, but also with other imputation methods, as shown in our experiments. Nonetheless, Figure 1.5 shows that MIM with mean imputation is

many orders of magnitude faster than using other imputation methods, yet usually results in about the same predictive performance. Therefore, MIM with mean imputation is a very effective and yet efficient way to treat missing values, especially under informative missingness when expensive imputation methods often bring little benefit.

## 1.8 Conclusion

When dealing with missing data, imputation with state-of-the-art imputers is often expensive. We show that using MIM in conjunction with mean imputation is an effective and efficient alternative, which we demonstrate via novel theory and comprehensive experimentation on synthetic data as well as real data with both synthetic and actual missing values. We additionally introduce Selective MIM (SMIM), a MIM-based preprocessor that discards uninformative missing indicators and is thus more stable than MIM on high-dimensional data sets. We show experimentally that adding MIM or SMIM helps achieve substantially better accuracy on data with informative missingness overall, and SMIM outperforms MIM on high-dimensional data. Future work might include researching theoretical guarantees on the use of imputation methods along with missing indicators, building on our theory.

## 1.9 Proofs

### 1.9.1 Proof of Theorem 1.3.1

Let $\boldsymbol{D} = [\tilde{Z}, R]^T$, then

$$\boldsymbol{D}\boldsymbol{D}^T = \begin{bmatrix} \tilde{Z} \\ R \end{bmatrix} \begin{bmatrix} \tilde{Z} & R \end{bmatrix} = \begin{bmatrix} \tilde{Z}^2 & 0 \\ 0 & R \end{bmatrix}. \tag{1.18}$$

as $R^2 = R$ and $\tilde{Z}R = 0$ by zero imputation (Assumption 1.3.1). Thus the finite sample OLS estimates $\hat{\beta}$ and $\hat{\gamma}$ are

$$\begin{aligned} \begin{bmatrix} \hat{\beta}_{MIM} \\ \hat{\gamma}_{MIM} \end{bmatrix} &= \mathbb{E}_n \left[\boldsymbol{D}\boldsymbol{D}^T\right]^{-1} \mathbb{E}_n \left[\boldsymbol{D}Y\right] & (1.19) \\ &= \begin{bmatrix} \mathbb{E}_n \left[\tilde{Z}^2\right]^{-1} \mathbb{E}_n[\tilde{Z}Y] \\ \mathbb{E}_n \left[R\right]^{-1} \mathbb{E}_n[RY] \end{bmatrix} & (1.20) \\ &= \begin{bmatrix} \hat{\beta} \\ \mathbb{E}_n[Y \mid R = 1] \end{bmatrix} & (1.21) \end{aligned}$$

The result for $\gamma$ in (1.21) holds because

$$\begin{aligned} \mathbb{E}_n \left[R\right]^{-1} \mathbb{E}_n[RY] &= \left(\frac{|\mathcal{M}|}{n}\right)^{-1} \frac{1}{n} \sum_{i \in \mathcal{M}} Y^{(i)} & (1.22) \\ &= \frac{1}{|\mathcal{M}|} \sum_{i \in \mathcal{M}} Y^{(i)} & (1.23) \\ &= \mathbb{E}_n[Y \mid R = 1] & (1.24) \end{aligned}$$

□

### 1.9.2 Proof of Theorem 1.3.2

**Part (a)** Let $\boldsymbol{D} = [\tilde{\boldsymbol{Z}}^T, \boldsymbol{R}^T]^T$ and $p_j = P(R = 1)$. We would like to show that $\mathbb{E}\left[\boldsymbol{D}\boldsymbol{D}^T\right]$ is a $2 \times 2$ block diagonal matrix. When $j \neq k$, $\mathbb{E}[\tilde{Z}_j R_k] = 0$ since $X_j$ and $R_k$ are independent under MCAR. If $j = k$, we cannot assume that $\tilde{Z}_j$ and $R_j$ are independent since they are directly dependent by construction. Nonetheless, by the law of total expectations we have

$$\begin{aligned}\mathbb{E}[\tilde{Z}_j R_j] &= \mathbb{E}[\tilde{Z}_j R_j \mid R_j = 1]p_j + \mathbb{E}[\tilde{Z}_j R_j \mid R_j = 0](1 - p_j) && (1.25)\\ &= \mathbb{E}[\tilde{Z}_j \mid R_j = 1]p_j && (1.26)\\ &= 0 && (1.27)\end{aligned}$$

since $\tilde{\boldsymbol{Z}}$ is imputed with 0. We have now shown that $\mathbb{E}[\tilde{Z}_j R_k] = 0$ for all $j, k$, showing as desired that $\mathbb{E}\left[\boldsymbol{D}\boldsymbol{D}^T\right]$ is block diagonal. Further, since $R$ is uninformative, we have for all $j$

$$\mathbb{E}[R_j Y] = \mathbb{E}[R_j]\mathbb{E}[Y] = 0 \tag{1.28}$$

using the centering of $Y$. Thus we have

$$\begin{aligned}\begin{bmatrix}\boldsymbol{\beta}^*_{MIM}\\ \boldsymbol{\gamma}^*_{MIM}\end{bmatrix} &= \left(\mathbb{E}[\boldsymbol{D}\boldsymbol{D}^T]\right)^{-1}\mathbb{E}[\boldsymbol{D}Y] && (1.29)\\ &= \begin{bmatrix}\mathbb{E}[\tilde{\boldsymbol{Z}}\tilde{\boldsymbol{Z}}^T]^{-1} & 0\\ 0 & \mathbb{E}[\boldsymbol{R}\boldsymbol{R}^T]^{-1}\end{bmatrix}\begin{bmatrix}\mathbb{E}[\tilde{\boldsymbol{Z}}Y]\\ 0\end{bmatrix} && (1.30)\\ &= \begin{bmatrix}\boldsymbol{\beta}^*\\ 0\end{bmatrix}, && (1.31)\end{aligned}$$

□

**Part (b)** For Part b we assume that $\boldsymbol{R}$ is centered, so

$$R_j = \begin{cases}1 - p_j & \text{w.p. } p_j\\ -p_j & \text{w.p. } 1 - p_j\end{cases}. \tag{1.32}$$

Like in the proof of (a), we want to show that $\mathbb{E}[\boldsymbol{D}\boldsymbol{D}^T]$ is $2 \times 2$ is $2 \times 2$ block diagonal. The only difference from the proof used in part (a) is that $\boldsymbol{R}$ is now centered. $\mathbb{E}[\tilde{Z}_j R_k] = 0$ when $j \neq k$ because $X_j$ and $R_k$ are still independent under the self-masking mechanism, and when $j = k$,

$$\begin{aligned}\mathbb{E}[\tilde{Z}_j R_j] &= \mathbb{E}[\tilde{Z}_j R_j \mid R_j = 1 - p_j]p_j + \mathbb{E}[\tilde{Z}_j R_j \mid R_j = -p_j](1 - p_j) && (1.33)\\ &= p_j(1 - p_j)\left(\mathbb{E}[\tilde{Z}_j \mid R_j = 1 - p_j] - \mathbb{E}[\tilde{Z}_j \mid R_j = -p_j]\right) && (1.34)\\ &= 0 && (1.35)\end{aligned}$$

Table 1.1: OpenML data sets used. Note that for volkert and christine, we remove several features that are either all 0 or are all 0 except a few rows, resulting in the dimensions listed.

| OpenML ID | Name | n | p | Task | n_classes |
|---|---|---|---|---|---|
| 23512 | higgs | 98050 | 28 | classification | 2 |
| 1458 | arcene | 200 | 10001 | classification | 2 |
| 41150 | miniboone | 130064 | 50 | classification | 2 |
| 41145 | philippine | 5832 | 309 | classification | 2 |
| 41142 | christine | 5418 | 1599 | classification | 2 |
| 1489 | phoneme | 5404 | 5 | classification | 2 |
| 6332 | bands | 540 | 16 | classification | 2 |
| 41166 | volkert | 58310 | 147 | classification | 10 |
| 40498 | wine | 4898 | 11 | classification | 7 |
| 41163 | dilbert | 10000 | 2000 | classification | 5 |
| 188 | eucalyptus | 736 | 9 | classification | 5 |
| 488 | college | 1161 | 6 | classification | 3 |
| 537 | housing | 20640 | 8 | regression | NA |
| 42705 | yolanda | 400000 | 100 | regression | NA |
| 507 | space | 3107 | 6 | regression | NA |
| 315 | crime | 1994 | 25 | regression | NA |

where $\mathbb{E}[\tilde{Z}_j \mid R_j = 1 - p_j] = 0$ because of 0 imputation and $\mathbb{E}[\tilde{Z}_j \mid R_j = -p_j] = 0$ because of centering.

Different from (a), though, $R_j \not\perp\!\!\!\perp Y$ now because both depend on $X_j$. We thus have

$$\mathbb{E}[R_j Y] = \mathbb{E}[Y R_j \mid R_j = 1 - p_j] p_j + \mathbb{E}[Y R_j \mid R_j = -p_j](1 - p_j) \tag{1.36}$$

$$= p_j(1 - p_j) \left( \mathbb{E}[Y \mid R_j = 1 - p_j] - \mathbb{E}[Y \mid R_j = -p_j] \right). \tag{1.37}$$

Lastly, $\mathbb{E}[R_j, R_k] = 0$ when $j = k$ since $R_j \perp\!\!\!\perp R_k$ under self-masking, and $\mathbb{E}[R_j^2] = p_j(1 - p_j)$. Putting this together, we have

$$\begin{bmatrix} \boldsymbol{\beta}^*_{MIM} \\ \boldsymbol{\gamma}^*_{MIM} \end{bmatrix} = \begin{bmatrix} \mathbb{E}[\tilde{\boldsymbol{Z}} \tilde{\boldsymbol{Z}}^T]^{-1} & 0 \\ 0 & \mathbb{E}[\boldsymbol{R}\boldsymbol{R}^T]^{-1} \end{bmatrix} \begin{bmatrix} \mathbb{E}[\tilde{\boldsymbol{Z}} Y] \\ \mathbb{E}[\boldsymbol{R} Y] \end{bmatrix} \tag{1.38}$$

$$= \begin{bmatrix} \beta_1^* \\ \vdots \\ \beta_p^* \\ \mathbb{E}[Y \mid R_1 = 1 - p_1] - \mathbb{E}[Y \mid R_1 = -p_1] \\ \vdots \\ \mathbb{E}[Y \mid R_p = 1 - p_p] - \mathbb{E}[Y \mid R_p = -p_p] \end{bmatrix}. \tag{1.39}$$

□

**Part (c)** The proof of part (c) starts by reordering the columns of $\boldsymbol{D} = [\tilde{\boldsymbol{Z}}^T, \boldsymbol{R}^T]^T$ based on the blocks $B_1, \ldots, B_d$, i.e.

$$\boldsymbol{D} \coloneqq [\boldsymbol{D}_{B_1}^T, \ldots, \boldsymbol{D}_{B_d}^T]^T \coloneqq [\tilde{\boldsymbol{Z}}_{B_1}^T, \boldsymbol{R}_{B_1}^T, \ldots, \tilde{\boldsymbol{Z}}_{B_d}^T, \boldsymbol{R}_{B_d}^T]^T. \tag{1.40}$$

Using the same logic as in the proof of part (b), we can conclude that $\mathbb{E}\left[\boldsymbol{D}\boldsymbol{D}^T\right]$ is $d \times d$ block diagonal, where $d$ is the number of blocks, and further for block $B_\ell$

$$\begin{bmatrix} \boldsymbol{\beta}^*_{MIM_{B_\ell}} \\ \boldsymbol{\gamma}^*_{MIM_{B_\ell}} \end{bmatrix} = \mathbb{E}[\boldsymbol{D}_{B_\ell}\boldsymbol{D}_{B_\ell}^T]^{-1}\mathbb{E}[\boldsymbol{D}_{B_\ell}Y]. \tag{1.41}$$

Recall from the proof of (b) that

$$\mathbb{E}[R_jY] = p_j(1-p_j)\left(\mathbb{E}[Y \mid R_j = 1-p_j] - \mathbb{E}[Y \mid R_j = -p_j]\right) \tag{1.42}$$

thus for block $B_\ell$

$$\mathbb{E}[\boldsymbol{D}_{B_\ell}\boldsymbol{D}_{B_\ell}^T]^{-1}\mathbb{E}[\boldsymbol{D}_{B_\ell}Y] \tag{1.43}$$

$$= \mathbb{E}[\boldsymbol{D}_{B_\ell}\boldsymbol{D}_{B_\ell}^T]^{-1}\begin{bmatrix} \mathbb{E}[\tilde{\boldsymbol{Z}}_{B_\ell}Y] \\ \mathbb{E}[Y \mid \boldsymbol{R}_{B_\ell} = 1-\boldsymbol{p}_{B_\ell}] - \mathbb{E}[Y \mid \boldsymbol{R}_{B_\ell} = -\boldsymbol{p}_{B_\ell}] \end{bmatrix} \tag{1.44}$$

where we've abused notation and used $\mathbb{E}[Y \mid \boldsymbol{R}_{B_\ell} = 1-\boldsymbol{p}_{B_\ell}]$ to mean the vector of $\mathbb{E}[Y \mid R_j = 1-p_j]$ for $j \in B_\ell$ and similarly for $\mathbb{E}[Y \mid \boldsymbol{R}_{B_\ell} = -\boldsymbol{p}_{B_\ell}]$. □

## 1.10 Additional Experiment Details

### 1.10.1 OpenML Data Sets

In Table 1.1 we show a description of the OpenML data sets used. The source data sets can easily by found by searching the IDs on OpenML. We chose these data sets to represent diversity in $n$, $p$, and outcome type. We also restrict to data sets with continuous features. For Figure 1.4, we only use data sets which have no missing values, since we want to be able to completely control the missing mechanism in the data. The listed values for $p$ are the number of features used in the data sets for our experiments, which in some cases is different than the listed number of features on OpenML. For example, there are several features in the volkert data set which all entirely 0, and so we remove these features.

### 1.10.2 MIMIC

The MIMIC-III data set [48] is a standard data set for building models on electronic health records (EHRs), and has been used by many papers to evaluate machine learning models [92, 28, 128, 8, 76, 24]. Due to its popularity, many tools have been developed to preprocess the raw MIMIC data into forms suitable for data science [35, 117, 83, 102]. We chose to use the mimic3-benchmark [35] to help preprocess our data, creating data sets for the mortality, length of stay (LOS), and phenotype prediction tasks described in Section 1.6.

For each of the above tasks, the mimic3-benchmark code gathers data from 17 clinical variables as features for the prediction. These 17 features are the same for each task. For each of the above tasks, the mimic3-benchmark code provides scripts for generating multivariate time series data, with 1 time series per feature per visit that covers a patient's data across their hospital stay. The benchmark code also provides an additional script to generate tabular data using feature engineering on the multivariate time series data to support their logistic regression baselines. Since we study tabular data, we use this additional preprocessing, and generate 1 feature for each clinical feature corresponding to the mean value across the observed components of the time series. When a time series has no observed time steps, we leave the tabular feature as missing.

# Chapter 2

# Interpretable Machine Learning for Survival Analysis

## 2.1 Introduction

Many healthcare problems require estimating the cumulative distribution function of a time-to-event random variable $T$ given a set of features $X$. For example, $T$ may represent the time until patient death, and the task is to estimate the mortality rate before time $t$ given patient features $X$ such as demographics, lab measurements, etc. Such time-to-event problems are often called *survival analysis* problems and have a long history in statistics [19].

While statistical approaches such as the Cox model [20] are useful when statistical inference is needed, recent machine learning (ML) approaches for survival analysis often provide better predictive accuracy [47, 62, 86, 42]. Unfortunately, most of these ML models are black-box, providing little to no explanation for their predictions. In healthcare settings, model interpretability is critical for several reasons. First, users (doctors and patients) are more likely to trust a model's predictions if the model can explain how each prediction is generated, including how each of the features contributed to the prediction [82]. Second, ML models in healthcare are often trained and validated on data from only one healthcare system, and without model interpretability, spurious signal learned by a model can remain hidden. For example, visitation by a priest can be highly correlated with mortality [22], and a black-box model could unknowingly rely heavily on this feature despite it potentially not being available in many healthcare systems. Model interpretability can also enable the discovery of new risk factors or previously unknown patterns in known risk factors, which is not possible with black-box models. For these reasons, black-box ML models for survival analysis are often greeted with trepidation in healthcare settings.

In contrast to black-box ML models, glass-box ML models are interpretable by design. Glass-box ML models can explain how each feature contributes to each prediction (local importance) as well as how each feature affects predictions globally on average (global importance and shape functions). In the context of supervised learning, recent literature shows that glass-box ML models can achieve performance comparable to black-box models, combining the benefits of black-box ML models and traditional statistical models [75, 81, 17, 44]. However, few glass-box ML models are available for survival analysis.

In this chapter, we introduce 3 glass-box approaches for survival analysis. The first method combines survival stacking and glass-box classification models for interpretable survival analysis [111]. The second method, called DyS (pronounced "dice"), is a neural additive model that focuses on combining feature selection and interpretability [113]. The third method, called DNAMite (pronounced "dynamite"), is a neural additive model with a specially designed embedding module that produces more accurate shape functions [110].

## 2.2 Background and Related Work

### 2.2.1 Survival Analysis

Given a set of features $X \in \mathbb{R}^p$ and time-to-event label $T \in \mathbb{R}^+$, the survival analysis problem is to estimate the conditional survival probability $P(T > t \mid X)$, or equivalently, the cumulative distribution function $P(T \leq t \mid X)$ which is often called the *cumulative incidence function* (CIF). To complicate the estimation, $T$ is commonly *censored* for some samples in the training dataset, so that only a lower bound on the time-to-event is known. Survival training data arrives in the form $(X, Z, \delta)$, where the observed event time $Z = \min(C, T)$ is the minimum of the true event time $T$ and the censoring time $C$, and the censor indicator $\delta = \mathbb{1}_{C>T}$ indicates whether a sample's event time is observed (1) or whether the event is censored (0).

Classical approaches to survival analysis often model the survival hazard function

$$\lambda(t \mid X) = \frac{p(t \mid X)}{P(T > t \mid X)} \tag{2.1}$$

where $p(t \mid X)$ is the conditional density function for $T$. For example, the Cox proportional hazards model [20] specifies a linear function for the log of the hazard function:

$$\lambda(t \mid X) = \lambda_o(t) \exp(\beta_1 X_1 + \cdots + \beta_p X_p) \tag{2.2}$$

By separating $\lambda(t \mid X)$ into a time-dependent intercept and a time-independent prediction, the Cox model enforces proportional hazards: for two subjects with features $X$ and $X'$, the ratio $\lambda(t|X)/\lambda(t|X')$ is independent of the time $t$. Newer ML models relax this assumption either by estimating the hazard function $\lambda(t \mid X)$ (or the CIF/survival function) at several times $t$ [62, 86, 42] or by estimating the parameters of a parametric distribution for $T$ [6].

### 2.2.2 Glass-Box Machine Learning

A glass-box machine learning (ML) model is a model that can produce accurate predictions while being transparent in how predictions are generated. Given the ambiguity around model "interpretability" [73], we consider a model to be a glass-box model if it can be completely described by a collection of simple plots. For example, a linear model is a glass-box model since it can be fully described by a single line plot for each feature. Such glass-box models are crucially different than black-box models pairs with post-hoc interpretability methods like SHAP [71] or LIME [87], as these approaches are inexact and prone to approximation bias

[56, 109].

Most literature on glass-box ML, including this paper, focuses on generalized additive models (GAMs). Given a set of $p$ features $X = (X_1, X_2, \ldots, X_p)^T$, a GAM $f$ has the form

$$f(X) = \beta_0 + f_1(X_1) + \cdots + f_p(X_p) \tag{2.3}$$

where each feature function $f_j$ is a function of only one feature $X_j$ and often called a *shape function*. GAMs can also be extended to higher order feature interactions; for example, a GAM with pairwise interactions (called a GA$^2$M) has the form

$$f(X) = \beta_0 + f_1(X_1) + \cdots + f_p(X_p) + \sum_{j \neq \ell} f_{j,\ell}(X_j, X_\ell). \tag{2.4}$$

In GA$^2$Ms, the individual feature functions are often called *main effects*, while the pairwise interaction functions are called *interaction effects*.

GAMs are glass-box models, since GAMs can be fully described by plotting each shape function. GAMs can also easily produce feature importances: the importance of feature $j$ is $\frac{1}{n}\sum_i |f_j(x_j)|$, which measures how much feature $j$ contributes to the prediction $f(x)$ on average across the training dataset. Further, GA$^2$Ms are also glass-box models under certain conditions. First, if the main effects and interaction effects are fit jointly, a purification procedure based on the functional ANOVA decomposition can be used to "purify" the main effects so that they maintain the entire marginal signal of each feature [63]. Alternatively, a GA$^2$M can first fit only main effects, freeze the main effects, compute the current prediction residual, and fit the interaction effects on these residuals.

## 2.2.3 Calibration

For binary classification problems with binary response $Y \in \{0, 1\}$, a *calibrated* model is a model that produces predictions $\hat{P}(Y = 1 \mid X)$ that correspond in a probabilistic sense to the true value of $P(Y = 1 \mid X)$. That is, a model is calibrated if for all sets $S = \{i : \hat{P}(Y^{(i)} = 1 \mid X^{(i)}) \approx \alpha\}$ we have

$$\alpha \approx \hat{P}(Y^{(i)} = 1 \mid X^{(i)}) \approx \sum_{i \in S} \frac{Y^{(i)}}{|S|}. \tag{2.5}$$

For binary classification models, calibration plots can check calibration by comparing prediction quantiles to true positive frequencies [74]. For survival analysis, models are considered calibrated if the CIF $P(T \leq t \mid X)$ is calibrated for all $t$. Unfortunately, censoring makes it harder to verify calibration, as the true event time $T$ is not known for all users. The standard approach to check calibration of survival models, which we follow in this paper, is to plot CIF prediction bins against Kaplan-Meier estimates for each bin; see [5] for more details.

## 2.2.4 Related Work

While explainable ML methods have a long history in healthcare [1], methods for explainable survival analysis have only recently been developed [58]. For post-hoc explanations, the SHAP package [71] supports post-hoc explanations for xgboost models trained with the Cox loss. Additionally, recent papers have introduced extensions to SHAP [3, 55] and LIME [54] for survival analysis. For glass-box models, traditional GAMs can be used with the Cox loss [37], but other survival losses requires outputting time-dependent prediction, which is made easier with NAMs due to parameter sharing when learning multiple outputs. As such, multiple NAMs have been proposed for survival analysis with various losses [121, 106, 113, 85]. Nonetheless, none of these previous survival NAMs address the issues of over-smooth shape functions and calibration, which we address with DNAMite.

## 2.3 Survival Stacking

Although few glass-box ML models have been developed specifically for survival analysis, numerous models have been designed for binary classification [37, 98, 40, 75, 2]. Survival stacking [21] offers a method to train binary classification models for survival analysis, thus enabling the application of these transparent classification models to survival analysis. Specifically, survival stacking creates a new binary classification dataset from a survival analysis dataset, such that probabilistic predictions on the new dataset correspond to hazard prediction on the original survival dataset.

We describe an improved version of survival stacking that better scales to large datasets [111]. This version can be readily combined with any glass-box binary classification model to create a glass-box survival model. We assume survival data $\{(X^{(i)}, T^{(i)}, \delta^{(i)})\}_{i=1}^{n}$ where $X$ is a vector of covariates, $T$ is the time when the event of interest occurs (e.g. getting heart failure), and $\delta$ is the censoring indicator, indicating whether, at time $T$, the patient reaches the event of interest or is lost to future observation before developing the condition. For each event time $t$ observed in the original data set, survival stacking adds all samples $X^{(i)}$ in the corresponding risk set $R(t) = \{i : T^{(i)} \geq t\}$ to a new "stacked" data set. For sample $i \in R(t)$, the corresponding binary label in the stacked data set is 0 if $T^{(i)} > t$, and 1 if $T^{(i)} = t$. Additionally, an extra covariate is added to the stacked data set representing the time $t$ which defines the risk set $R(t)$. Survival stacking works because training a binary classifier on the survival stacked data estimates the hazard function $\lambda(t \mid X)$ An example of survival stacking on a smaller data set can be found in [21].

We use an improved version of survival stacking in our pipeline, which is outlined in Algorithm 2.1. Specifically, we make two modifications to survival stacking as in [21] to better scale to large survival data sets. First, instead of defining a one-hot-encoded categorical variable to represent time, we define a single continuous time feature. This choice reduces the number of features, aiding computational efficiency and interpretability. Second, stacking increases the number of samples quadratically, potentially creating computational hardships as well as a severe class imbalance in the case of a high censoring rate. We perform random undersampling of the majority class samples $\{i : T^{(i)} > t\} \subset R(t)$ to mitigate this issue.

### Survival Stacking Prediction

After training a classification model $f$ on survival stacked data, we must convert the model's predicted probabilities to survival curves for patients during inference. In [21], the survival curve $S(t \mid X) = P(T > t \mid X)$ is estimated using

$$\hat{S}(t \mid X) = \prod_{t_k \leq t} \Big(1 - f(X \parallel t_k)\Big). \tag{2.6}$$

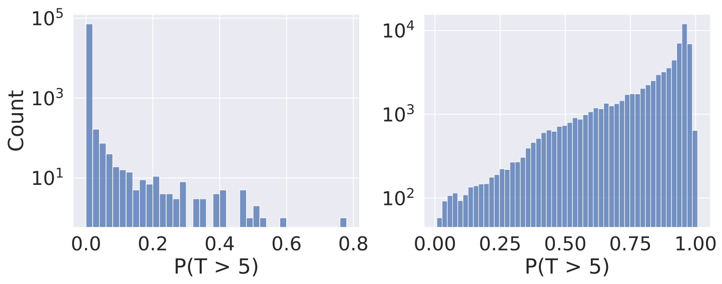


Figure 2.1: Distribution of predicted survival probabilities $S(t \mid X) = P(T > t)$ at $t = 5$ across test patients. The left plot shows the survival probabilities using Equation 2.6 as in [21], while the right plot shows Equation 2.9. Our survival prediction method gives a reasonable distribution, while the method from [21] is incorrectly skewed torwards 0.

where $\|$ represents concatenation to add the extra time covariate from survival stacking. The motivation for Equation 2.6 is that if the time variable $T$ is assumed to be discrete, taking on only the observed times $t_1, \ldots, t_k$ in the training set, then, assuming without loss of generality $t_1 \leq t_2 \leq \cdots \leq t_k \leq t$, $S(t \mid X)$ can be written as the product of conditional survival probabilities up until time $t$:

$$S(t \mid X) = P(T > t \mid X) = \prod_{t_k \leq t} \Big(1 - \lambda(t_k \mid X)\Big) \tag{2.7}$$

see [101]. This motivates Equation 2.6 since the binary classifier $f$ estimates the hazard function $\lambda(t \mid X)$ with survival stacking. However, the estimate in Equation 2.6 can become unstable in large sample, as demonstrated in Figure 2.1. Further, we use a continuous time feature in our survival stacking algorithm, implying that this discrete time assumption may not be suitable. Thus, we propose a different method for predicting the survival curve, which is summarized in Algorithm 2.2. We first predict the cumulative hazard function $\Lambda(t \mid X) = \int_0^t \lambda(s \mid X) ds$ using Monte Carlo integration at $n$ uniform continuously sampled times $t_1, \ldots, t_n \leq t$:

$$\hat{\Lambda}(t \mid X) = \frac{t}{n} \sum_{i=1}^{n} f(X \parallel t_i). \tag{2.8}$$

After estimating $\Lambda(t \mid X)$, we can naturally estimate $S(t \mid X)$ using

$$\hat{S}(t \mid X) = \exp\left(-\hat{\Lambda}(t \mid X)\right). \tag{2.9}$$

Using this estimator is possible since we used a continuous time feature in survival stacking, which allows for such Monte Carlo integration.

## 2.3.1 Comparison to Glass-Box Survival Models

The main advantage of survival stacking is that it is a general-purpose tool that can be readily combined with any binary classification model. However, it has limitations compared to glass-box models like DyS and DNAMite can are specifically designed for survival analysis. First, the feature importances and shape functions from a glass-box model trained with survival stacking describe contributions to the hazard function. In comparison, glass-box survival analysis models can be designed to yield interpretations to more intuitive

**Algorithm 2.1** Survival Stacking With Subsampling

```
1: Input: Survival data $(X_1, T_1, Y_1), \ldots,$
          $(X_n, T_n, Y_n)$, sampling ratio $\gamma$.
2: Output: Classification data.
3: event_times $\leftarrow \{T_i : Y_i = 1\}$, samples $\leftarrow [\,]$.
4: for $t$ in event_times do
5:     samples += $\{(X_i \parallel T_i, 1) : T_i = t, Y_i = 1\}$.
6:     risk_set = uniform random sample with
                  probability $\gamma$ from $\{i : T_i > t\}$.
7:     samples += $\{(X_i \parallel T_i, 0) : i \in$ risk_set$\}$.
8: end for
9: return samples.
```

**Algorithm 2.2** Survival Prediction

```
1: Input: Fitted classification model $f$, survival data test sample $X$, prediction time $t$.
2: Output: Estimated survival probability
          $\hat{S}(t \mid X) = P(T > t \mid X)$.
3: Sample $t_1, \ldots, t_n$ uniformly from $(0, t]$.
4: Estimate CHF via Monte Carlo integration:
          $\hat{\Lambda}(t \mid X) = \frac{t}{n} \sum_{i=1}^{n} f(X \parallel t_i)$.
5: return $\hat{S}(t \mid X) = \exp(-\hat{\Lambda}(t \mid X))$.
```

quantities, such as time-dependent survival probability. Second, even with the aforementioned improvements to survival stacking, scaling to large datasets is inefficient, especially if both $n$ and $p$ are large.

## 2.4 DyS: Feature-Sparse Survival Analysis

DyS (pronounced "dice") is a glass-box ML model for survival analysis [113]. DyS is trained using a ranked probability score (RPS) loss function that directly optimizes the survival predictions, leading to better discrimination [50, 6]. Additionally, DyS can perform feature selection during the model fitting process, both on the main effects and on the interaction terms.

### 2.4.1 Model Architecture

The model architecture for DyS is summarized in Figure 2.2. Following previous glass-box models, DyS is a generalized additive model with interactions, or a GA$^2$M model. We choose to parameterize each shape function in DyS using neural networks, specifically MLPs, making DyS a neural additive model (NAM) with interactions, or a NA$^2$M.

DyS uses a discrete-time model, summarized in Algorithm 2.3, to generate survival predictions. We assume $T$ is discrete with finite support at $K$ distinct times: $T \in \{t_1, t_2, \ldots, t_K\}$. Under this model, DyS produces $K$-dimensional outputs $f(X_j), f(X_j, X_\ell) \in \mathbb{R}^K$, for each feature $X_j$ and interaction $(X_j, X_\ell)$. The global $K$-dimensional output $f(X)$ is obtained by summing the outputs from all main effects and interactions. Next, a softmax layer computes probability mass estimates $\hat{P}(T = t_k \mid X)$ for $k = 1, \ldots, K$. The probability of survival to time $t_k$, $\hat{S}(t_k \mid X)$, is one minus the sum of the probability mass estimates at earlier times.

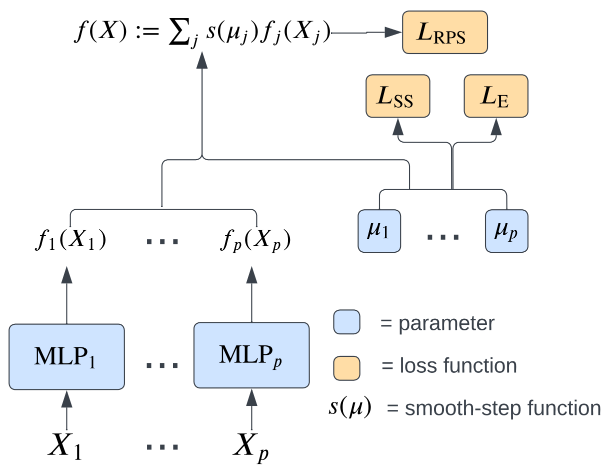


Figure 2.2: Summary of the architecture of DyS. For simplicity, the interaction effects are not shown. When feature sparsity is desired, the $\mu_j$ parameters are learned such that a subset of $s(\mu_j), j = 1, \ldots, p$ are equal to 0, preventing the corresponding features from influencing the predictions.

**Algorithm 2.3** DyS Survival Predictions

```
1: Input: Evaluation times $t_1, \ldots, t_K$; feature logits $f_j(X_j) \in \mathbb{R}^K$; interaction logits $f_{j,\ell}(X_j, X_\ell) \in \mathbb{R}^K$.
2: Output: Survival predictions $\hat{S}(t_k \mid X),\ k = 1, \ldots, K$.
3: $f(X) \leftarrow \sum_j f_j(X_j) + \sum_{j,\ell} f_{j,\ell}(X_j, X_\ell)$.
4: $\{\hat{P}(T = t_k \mid X)\}_{k=1}^K \leftarrow \text{softmax}(f(X))$.
5: for $k = 1$ to $K$ do
6:     $\hat{S}(t_k \mid X) \leftarrow 1 - \sum_{t \le t_k} \hat{P}(T = t \mid X)$.
7: end for
8: return $\hat{S}(t_k \mid X),\ k = 1, \ldots, K$.
```

**Interpretation Plots**

DyS can be fully summarized by feature (and feature interaction) impact plots, one for each evaluation time. Specifically, the feature impact plot for feature $j$ shows the logits of $P(T = t \mid X_j)$ at each evaluation time $t$. Each feature impact plot is normalized to sum to zero across the training data in order to avoid identifiability issues. Given a set of features $X$, the model's survival predictions $\hat{S}(t \mid X)$ can be obtained directly from these plots by following Algorithm 2.3. DyS's feature impact plots are critically different than partial dependence plots [32], as DyS's feature impact plots directly describe the model behavior using only the model structure.

DyS can also compute the importance of each feature $j$ at each time $t$ by averaging the absolute value of the logits of feature $j$ and time $t$ across all training samples. If global feature importances are desired (i.e. independent of evaluation time), the importances at each evaluation time can further be averaged.

## 2.4.2 Loss Function

To train survival models, we must map the continuous prediction function $f(X)$ to a differentiable loss function. Choosing an appropriate loss function for survival analysis is critical. While traditional supervised machine learning problems often use a standard loss functions (MSE for regression, cross-entropy for classification), there is no standard loss function for survival analysis.

The most commonly used loss function in survival analysis is the Cox proportional hazards loss:

$$-\prod_{i:\delta^{(i)}=1} \frac{\exp(f(X^{(i)}))}{\sum_{j:T^{(j)}\geq T^{(i)}} \exp(f(X^{(j)}))}, \tag{2.10}$$

where $(X^{(i)}, T^{(i)}, \delta^{(i)})$ is the $i^{\text{th}}$ sample in the training dataset. This loss was originally derived as a negative partial likelihood function for the Cox model [20] (in the Cox model, $f(X) = X^T\beta$). Intuitively, the loss works by maximizing the risk $\exp(f(X^{(i)}))$ for uncensored samples $i$ at their event-times $T^{(i)}$, relative to the other samples $j : T^{(j)} \geq T^{(i)}$ still at risk at time $T_i$. In practice, the log of Equation 2.10 is usually optimized for numerical stability.

There are two major problems with the Cox loss. First, the Cox loss enforces proportional hazards, as illustrated in Equation 2.10. This property asserts that if the model predicts a higher risk for sample $i$ than sample $j$ at time $t$, it also predicts a higher risk for sample $i$ at all other times, which may be unreasonable. For example, consider mortality prediction: a childhood cancer patient is at much higher risk of mortality than a typical 60 year old patient, but if the child lives into adulthood, their risk may drop relative to the older patient. We revisit this example empirically in Section **??**. Second, the Cox loss (or anything similar, e.g. the time-dependent Cox loss [103]) takes as input risk predictions rather than survival predictions. Thus, models that use the Cox loss must transform the model predictions in order to obtain survival predictions, which are usually the quantity of interest.

For these reasons, we choose the Ranked Probability Score (RPS) loss function [50] instead of a Cox-based loss. First, DyS follows Algorithm 2.3 to generate a survival curve estimate $\hat{S}(t \mid X)$ from the prediction function $f(X)$. Then the RPS loss is calculated as

$$L_{\text{RPS}}(\hat{S}, T, \delta) = \sum_{t<T} (1 - \hat{S}(t \mid X))^2 + \delta \sum_{t\geq T} \hat{S}(t \mid X)^2 \tag{2.11}$$

for a finite set of evaluation times $t \in (0, \max(T))$. The RPS loss is a discrete-time version of the Continuous Ranked Probability Score [29, 6] originally proposed for time series forecasting. The first term in $L_{\text{RPS}}$ maximizes survival for all samples before their respective event time, while the second term in $L_{\text{RPS}}$ minimizes survival for all uncensored samples after their event time. Compared to Cox-style losses, the RPS loss directly optimizes survival predictions, the usual quantity of interest.

### 2.4.3 Feature Sparsity

To obtain a feature-sparse model, inspired by [45], we introduce binary gates into our GA$^2$M model:

$$f(X) = \sum_j f_i(X_j) z_j + \sum_{j,\ell} f_{j,\ell}(X_j, X_\ell) z_{j,\ell}. \tag{2.12}$$

Each binary gate $z_j, z_{j,\ell} \in \{0, 1\}$ controls whether or not each feature/interaction is used by the final model. The binary gates are learned as parameters in the model alongside the shape functions $f_j, f_{j,\ell}$. Given the discrete nature of the binary gates, we follow [45] and replace the binary gates with smooth-step functions

$s(\mu_j), s(\mu_{j,\ell}) \in [0,1]$ with real-valued parameters $\mu_j, \mu_{j,\ell} \in \mathbb{R}$ [38]. The smooth-step function is defined as

$$s(\mu;\gamma) = \begin{cases} 0 & \text{if } \mu \leq -\gamma/2 \\ -\frac{2}{\gamma^3}\mu^3 + \frac{3}{2\gamma}\mu + \frac{1}{2} & \text{if } -\gamma/2 \leq \mu \leq \gamma/2 \\ 1 & \text{if } \mu \geq \gamma/2 \end{cases} \tag{2.13}$$

The smooth-step function is a continuous function with range $[0,1]$, but unlike other such functions like the sigmoid function, the smooth-step function can actually reach 0, producing an exactly sparse model. The resulting GA$^2$M model is

$$f(X) = \sum_j f_j(X_j)s(\mu_j) + \sum_{j,\ell} f_{j,\ell}(X_j, X_\ell)s(\mu_{j,\ell}). \tag{2.14}$$

To induce feature sparsity, the smooth-step parameters $\mu_j, \mu_{j,\ell}$ are learned via the RPS loss $L_{RPS}$ along with the sparsity regularizer $L_{\mathrm{SS}}(\mu)$:

$$L_{\mathrm{SS}}(\mu) = \lambda \left( \sum_j s(\mu_j) + \alpha \sum_{j,\ell} s(\mu_{j,\ell}) \right). \tag{2.15}$$

Last, to control the rate at which the smooth-step functions converge to $\{0,1\}$, we add the entropy regularizer $L_{\mathrm{E}}(\mu)$:

$$L_{\mathrm{E}}(\mu) = \tau \left( \sum_j \Omega(s(\mu_j)) + \sum_{j,\ell} \Omega(s(\mu_{j,\ell})) \right), \tag{2.16}$$

$$\Omega(x) = -\Big(x\log(x) + (1-x)\log(1-x)\Big) \tag{2.17}$$

where $\Omega(x)$ explicitly encourages each $s(\mu_j), s(\mu_{j,\ell})$ to converge to 0 or 1. DyS is trained by minimizing $L_{RPS} + L_{SS} + L_E$ with respect to the trainable parameters in each shape function as well as the smooth-step parameters.

#### Preset Feature Budget

Occasionally, a user may want a model that uses a fixed number of features. This can be achieved by finding a hyperparameter $\lambda$ that yields the desired number of non-zero features. One algorithmic way to select such a hyperparameter is *bisection*, which uses a binary search algorithm to converge to a feasible hyperparameter value.

### 2.4.4 Two-Stage Fitting

Training GA$^2$Ms is computationally and memory intensive when the number of features in a dataset is large. For example, a dataset with $p = 1,000$ features contains $\sim 500,000$ possible interactions. Previous literature has suggested an initial round of *interaction screening* in order to reduce the number of interactions in the model: EBM [69] uses an efficient plane cutting algorithm, while Grand-Slamin' [45] fits shallow decision trees for all possible interactions to find the most promising interactions. Unfortunately, we found even these efficient methods to be too inefficient when both $n$ and $p$ are large. For example, on our heart failure dataset with training data of size $(n, p) = (537836, 2410)$, the estimated run time for fitting a single decision tree on all possible interactions (as in [45]) is over 100 hours.

Instead, DyS takes advantage of the feature sparsity in the model. We first fit only the main effects of the model. The fitting procedure naturally chooses an active subset of the main effects by learning the smooth-step function parameters. Then, we freeze the main effects and fit interaction effects only for interactions between two active main effects. This two-stage fitting approach, summarized in Algorithm 2.4, has both computational and interpretability benefits: fitting main effects first ensures that each main effect captures the entire available signal and does not leak into the interaction shape function. Hence, the shape function for each main effect represents the "pure" effect of each feature [63].

**Algorithm 2.4** DyS Two-Stage Fitting

```
1: Input: Survival dataset $D$.
2: Output: Fitted DyS model $f$.
3: Fit main effects of DyS model $f$ on $D$ using $L_{\text{RPS}} + L_{\text{SS}} + L_{\text{E}}$.
4: Freeze existing parameters of $f$.
5: Gather candidate interactions $(X_j, X_\ell)$ such that $s(\mu_j) > 0$ and $s(\mu_\ell) > 0$ in $f$.
6: Fit parameters in $f$ for candidate interactions.
7: return Fitted DyS model $f$.
```

## 2.5 DNAMite: Discretized Glass-Box Survival Models

DNAMite (pronounced "dynamite"), like DyS, is a GA$^2$M, i.e. a generalized additive model with pairwise feature interactions [110]. However, DNAMite makes several improvements compared to DyS. Most noteworthy, DNAMite discretizes all features into feature bins, and uses a specialized embedding module to learn an embedding for each unique bin. This enables DNAMite to learn shape functions with a controllable level of smoothness. Additionally, DNAMite optimizes the Inverse Probability of Censoring Weighting (IPCW) loss, which is a proper scoring loss for survival analysis and therefore improves model calibration compared to the RPS loss used in DyS.

### 2.5.1 Model Components

DNAMite estimates the CIF $P(T \leq t \mid X)$ by predicting $\hat{P}(T \leq t \mid X)$ at $K$ evaluation times $t = t_1, \ldots, t_K$. To generate these predictions, DNAMite produces a $K$-dimensional vector for each feature and interaction, which are summed and passed through a sigmoid function to obtain $\hat{P}(T \leq t \mid X)$ for each $t$. Each feature/interaction function consists of an embedding module followed by a multi-layer perception (MLP). We use neural networks for each function (instead of splines or boosted decision trees) to facilitate the use of parameter sharing to seamlessly allow each function to have a multivariate output. The embedding module is imperative to DNAMite, as it is crucial for allowing the model to accurately estimate shape functions (see Figure 2.4). The following two subsections detail the key components of the embedding module.

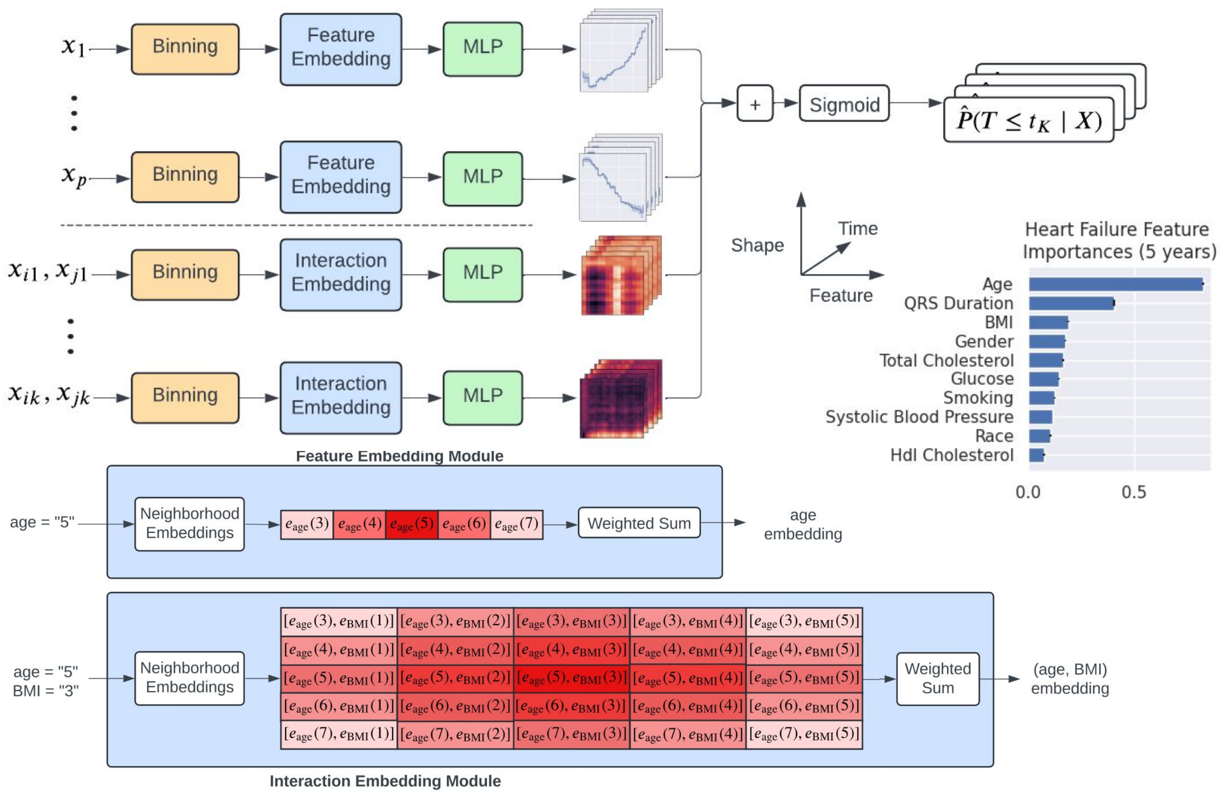


Figure 2.3: Overview of DNAMite. (Top Left) Each of the $p$ feature and $k$ interaction shape functions consisting of an embedding module followed by a multi-layer perceptron (MLP). The final prediction sums the $K$-dimensional output of each feature/interaction function followed by a sigmoid activation to output $K$ CIF estimates. (Bottom) Main effect and interaction embedding modules. Interaction embeddings concatenate individual feature embeddings. Final embeddings are computed as weighted sums of the embeddings from neighboring feature/interactions. (Right) Example feature importance scores for DNAMite from the heart failure dataset.

### Discretization

Unlike previous survival NAMs, DNAMite discretizes both continuous and categorical features. To discretize a continuous feature into $b$ bins, DNAMite uses feature quantiles to define $b-1$ cut points that split the data into $b$ unique bins. One additional bin represents missing values. After defining the feature bins, DNAMite replaces each original feature value with the corresponding bin index, ranging from $0, 1, \ldots, b$. DNAMite learns a different embedding for each possible feature bin. For categorical features, each unique level gets its own embedding, with one extra bin again for missing values. For interactions, DNAMite discretizes and embeds each features in the interaction separately and concatenates the two embeddings.

Why discretize a continuous feature? Discretization loses information, yet counterintuitively, recent evidence suggests this loss of granularity can improve predictive performance for neural networks [4, 31, 43]. One explanation is that discretization, when combined with embedding layers, allows the model to learn embeddings specialized for specific feature ranges, which is much more challenging when embedding continuous features without discretization. Discretization also allows for seamless handling of special feature values.

### Embeddings

While most NAMs learn overly smooth feature functions, explicit feature discretization causes the opposite problem: learned feature functions can overfit each individual bin, causing jaggedness (as shown in Figures 2.4 and 2.6). This jaggedness occurs because the embeddings for neighboring feature bins are learned independently, ignoring ordinality. Such ignorance is different from tree-based models, which also discretize continuous features but make feature splits using ordinal information.

To overcome this limitation, DNAMite learns embeddings for each feature bin that take advantage of the ordering of each continuous feature. The embedding module for individual features and interactions is visualized in Figure 2.3. First, each feature gets a standard embedding module: a lookup table is maintained with $d$-dimensional embeddings for each unique feature bin. Final embeddings are computed as weighted sums of the embeddings from neighboring feature bins, with weights determined by a kernel function that depends on the number of bins between a reference bin and neighbor bin. A kernel hyperparameter $\gamma$ controls the smoothness of the learned embeddings. For full details on the embedding module for main effects, see Algorithm 2.5.

**Algorithm 2.5** Feature Embeddings

```
1: **Input**: feature value $x \in \mathbb{R}$, binning function $b : \mathbb{R} \to \mathbb{Z}^+$, embedding function $e : \mathbb{Z}^+ \to \mathbb{R}^d$
   kernel strength $\gamma$, kernel width $k$, max bin size $B$.
2: Replace $x$ with bin index $b(x)$.
3: kernel_weights ← [exp $\left(-\frac{(x-z)^2}{2\gamma}\right)$ for $z \in [-k, k] \cap \mathbb{Z}$]
4: neighbor_bins ← [$x + z$ for $z \in [-k, k] \cap \mathbb{Z}$].
5: Remove neighbors where neighbor_bins$_i \leq 0$ or neighbor_bins$_i > B$.
6: embeds ← [$e$(index) for index in neighbor_bins]
7:
8: **return** $\sum_i$ kernel_weights$_i \cdot$ embeds$_i$.
```

## 2.5.2 Interpretability

DNAMite can produce feature importances and estimated shape functions for the CIF at *each evaluation time* used during model training. These granular interpretations are critical to capture important patterns in survival data. For example, for survival datasets where the proportional hazards assumption is not reasonable, time-dependent interpretations allow DNAMite to capture signal that Cox-based models would ignore [113]. While DyS [113] also produces shape functions for each feature at each time, DyS visualizes probability mass estimates $\hat{P}(T = t \mid X)$, which are more difficult to interpret than DNAMite's CIF estimates $P(T \leq t \mid X)$. Importantly, it is not possible to convert DyS's interpretations to CIF-based interpretations because the final sigmoid activation is applied to the sum of the features; this final nonlinear transformation confounds any additive interpretation via shape functions. Lastly, PseudoNAM can also output time-depednent interpretations for the CIF, but PseudoNAM involved computing pseudo-values which is very computationally slow for large dataset (too slow to complete in less than 5 hours on 2 out of 5 of our experiment datasets in Table 2.1).

### 2.5.3 Calibration

As discussed in Section 2.2.3, glass-box models must be well-calibrated, as post-hoc calibration confounds the interpretation of model outputs. Unlike previous interpretable survival models, DNAMite produces calibrated predictions without post-hoc calibration. To produce calibrated predictions, DNAMite is trained with the Inverse Probability of Censoring Weighting (IPCW) loss. Given CIF predictions $\hat{p}(t_k) = \hat{P}(T \leq t_k \mid X)$, the IPCW loss is given by:

$$\sum_{i=1}^{n}\sum_{k=1}^{K} \frac{\mathbb{1}_{Z_i > t_k}\hat{p}(t_k)^2}{\hat{P}(C > t_k)} + \frac{\mathbb{1}_{Z_i \leq t_k, \delta_i = 1}(1-\hat{p}(t_k))^2}{\hat{P}(C > Z_i)} \tag{2.18}$$

The IPCW loss is a proper scoring loss under the assumption that censoring is independent of features, i.e. $C \perp\!\!\!\perp X$ [88], which means that optimizing the IPCW loss learns asymptotically calibrated predictions. To estimate the censoring probabilities $\hat{P}(C > t_k)$, we fit a Kaplan-Meier estimator before training DNAMite. If there is reason to believe that $C \not\perp\!\!\!\perp T$, then as described in [88] the Kaplan-Meier estimate $\hat{P}(C > t_k)$ can be replaced with any calibrated survival estimate $\hat{P}(C > t_k \mid X)$, e.g. a Cox regression model followed by post-hoc calibration [5].

### 2.5.4 Training Specifications

The training algorithm for DNAMite is fully described in Algorithm **??**. We highlight a few aspects of the training procedure:

**Identification** GAMs such as DNAMite suffer from unidentifiability without imposing proper constraints [15, 70]. For example, for two features $X_j$ and $X_\ell$, shifting $f_j(X_j) \rightarrow f_j(X_j) + 1$ and simultaneously shifting $f_\ell(X_\ell) \rightarrow f_\ell(X_\ell) - 1$ results in the same predictions but different shape functions. DNAMite ensures identifiability by constraining the predictions of every shape function to sum to 0 across the training dataset. See Algorithm 2.6.

**Algorithm 2.6** Compute Intercept

```
1: Input: DNAMite model $f$, train data $D = \{(X^{(i)}, Y^{(i)})\}_{i=1}^n$.
2: $\beta_0 \leftarrow 0$.
3: for $j = 1, \dots p$ do
4:     $f_j \leftarrow f_j - \frac{1}{n}\sum_{i=1}^n f_j(X_j^{(i)})$
5:     $\beta_0 \leftarrow \beta_0 + \frac{1}{n}\sum_{i=1}^n f_j(X_j^{(i)})$
6: end for
7: for pairs $j, \ell$ in $f$ do
8:     $f_{j,\ell} \leftarrow f_{j,\ell} - \frac{1}{n}\sum_{i=1}^n f_{j,\ell}(X_j^{(i)}, X_\ell^{(i)})$
9:     $\beta_0 \leftarrow \beta_0 + \frac{1}{n}\sum_{i=1}^n f_{j,\ell}(X_j^{(i)}, X_\ell^{(i)})$
10: end for
11: Add $\beta_0$ to $f$
12:
13: return $f$.
```

**Two-Stage Training** DNAMite learns main effects and interaction effects in two stages to avoid the purification step described in Section 2.2.2. Specifically, main effect terms are first learned via backprop until convergence. Then the main effect weights are frozen and the interaction terms are learned through backprop

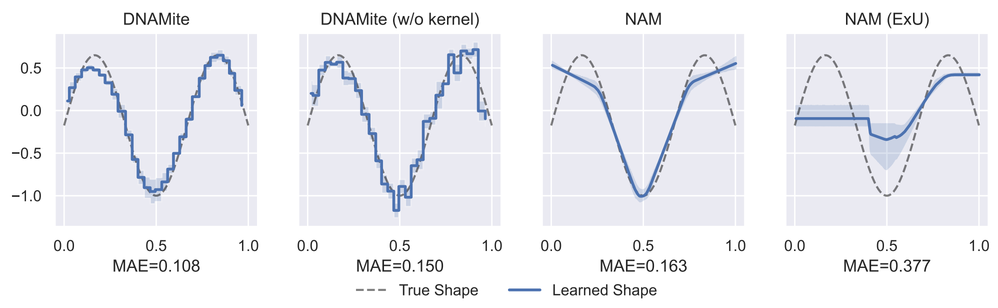


Figure 2.4: Comparison of model performance in learning one feature's shape function for synthetic data. DNAMite captures the true shape more accurately than competing methods.

on the prediction function. As the interaction effects sum embeddings for each of the interacting features, the (frozen) main effect embeddings are used to warm start the interaction embedding module.

**Confidence intervals.** Confidence intervals for shape functions are critical help diagnose variability in each feature's contribution. DNAMite uses cross-validation to generate confidence intervals for shape functions. DNAmite splits the training data into $B$ different train/validation splits and trains one model on each split. The final prediction is the mean prediction from each of the $B$ DNAMite models, and error bars for the shape functions are generated by computing confidence intervals for the mean at each unique bin across the $B$ models.

## 2.6 Experiments

### 2.6.1 Synthetic Data

Since true feature shape functions are unknown in real-world data, we generate synthetic survival analysis data to assess the ability of DNAMite and other glass-box models to accurately estimate shape functions. To evaluate models across both simple and challenging shapes, we generate synthetic survival data where each feature's true shape functions is defined by piece-wise or continuous functions with varying levels of jaggedness.

We compare DNAMite to three baseline models. First, to assess the impact of kernel smoothing on the learned shape functions, we set the smoothing parameter in DNAMite to $\gamma = 0$, which we call DNAMite (w/o kernel). Second, we remove the embedding module entirely from DNAMite and call the resulting model NAM. Third, we train NAM with ExU activations in an attempt to improve the resulting shape functions [2].

Figure 2.4 illustrates the performance of DNAMite and other NAMs on one of the synthetic features We evaluate the ability of each model to capture shape functions by calculating the mean absolute error (MAE) between the predicted and true shape functions. DNAMite consistently captures feature shapes more accurately, showing lower MAE across all features. DNAMite handles complex and jagged functions far better than NAM, while maintaining competitive performance on simple feature shapes. Without kernel smoothing, DNAMite tends to learn overly jagged shape functions, demonstrating the necessity of kernel smoothing for avoiding overfitting. Surprisingly, despite producing very different shape functions, DNAMite achieves

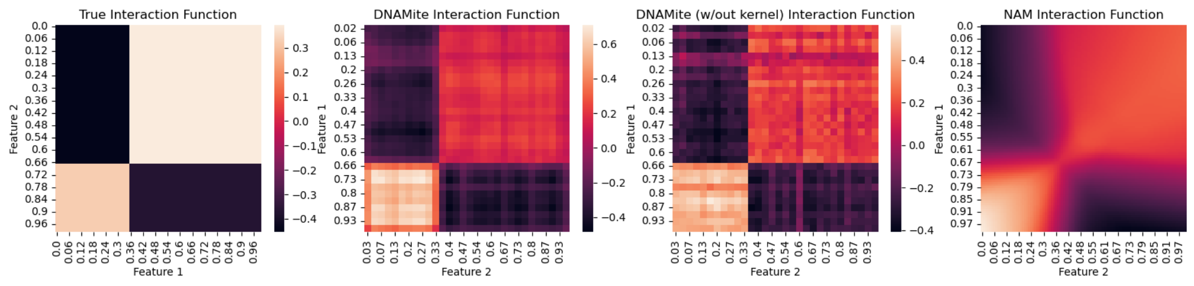


Figure 2.5: Comparison of models when learning a true shape function on synthetic data.

Table 2.1: Mean Time-Dependent AUC for DNAMite versus baselines. Models are run for 5 trials (mean ± standard deviation shown), each with a different random seed and train/test split. DNAMite performs similar to the best model on each dataset. Results within one standard deviation of the top performer for each dataset are bold. † indicates a black-box model, and * indicates termination due to high runtime ≥ 5 hours.

| dataset | flchain | metabric | support | unos | heart_failure |
|---|---|---|---|---|---|
| RSF† | **0.955 ± 0.003** | 0.730 ± 0.027 | **0.831 ± 0.006** | * | * |
| DeepHit† | **0.955 ± 0.002** | 0.714 ± 0.022 | 0.752 ± 0.006 | 0.753 ± 0.004 | 0.836 ± 0.004 |
| SATransformer† | **0.955 ± 0.002** | 0.723 ± 0.026 | 0.826 ± 0.008 | 0.769 ± 0.004 | **0.843 ± 0.005** |
| DRSA† | 0.939 ± 0.010 | 0.727 ± 0.027 | **0.835 ± 0.009** | 0.758 ± 0.004 | 0.809 ± 0.008 |
| CoxPH | 0.954 ± 0.002 | 0.707 ± 0.020 | 0.815 ± 0.008 | 0.692 ± 0.003 | 0.827 ± 0.004 |
| AFT | 0.954 ± 0.002 | 0.707 ± 0.020 | 0.812 ± 0.008 | 0.689 ± 0.003 | 0.827 ± 0.003 |
| mgcv | 0.905 ± 0.006 | 0.695 ± 0.015 | 0.812 ± 0.010 | 0.643 ± 0.005 | 0.514 ± 0.007 |
| PseudoNAM | **0.955 ± 0.002** | 0.712 ± 0.016 | **0.834 ± 0.008** | * | * |
| CoxNAM | 0.928 ± 0.021 | 0.729 ± 0.025 | 0.819 ± 0.008 | 0.752 ± 0.003 | 0.779 ± 0.134 |
| DyS | **0.957 ± 0.002** | 0.730 ± 0.023 | **0.838 ± 0.007** | 0.724 ± 0.003 | 0.829 ± 0.008 |
| DNAMite | 0.950 ± 0.002 | **0.756 ± 0.015** | **0.834 ± 0.008** | **0.779 ± 0.006** | **0.841 ± 0.002** |

similar predictive performance with and without kernel smoothing. This result demonstrates that kernel smoothing is necessary to avoid overfitting with respect to true shape functions, but not with respect to predictive performance. In contrast, NAM learns shape functions that are overly simple and smooth, missing important nuances of feature shapes as a result. Additionally, contrary to suggestions from [2], we find that using ExU magnifies this issue, resulting in even less accurate shape functions.

Lastly, we also compare DNAMite to competing methods for learning true interaction functions on synthetic data. We generate an interaction term for two features by defining thresholds that split the feature pair into four regions and assigning a different score to each region. The scores are normalized to have a mean effect of 0 so that interaction effects so not bleed into true main effects. Figure 2.5 shows that DNAMite can more accurately learn interactions compared to other models.

### 2.6.2 Real Data Benchmark

We now evaluate DNAMite and DyS on real-world survival data, comparing to the following other glass-box and black-box ML models.

- **CoxPH**: linear Cox model [20].
- **AFT**: Accelerated Failure Time model, another linear model [118].
- **mgcv**:

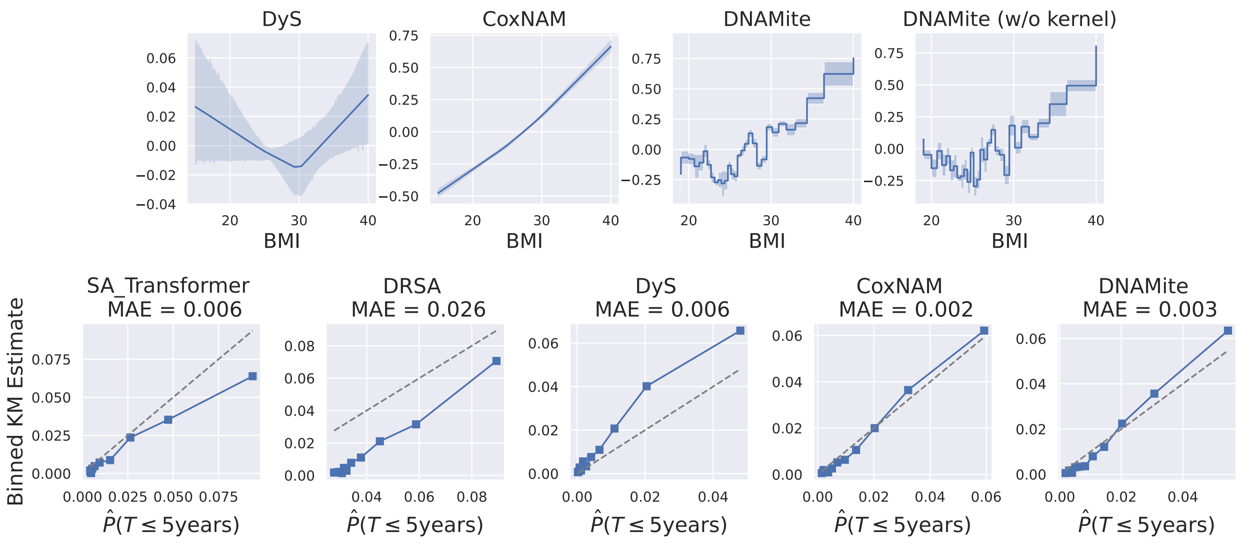


Figure 2.6: (Top) Shape functions for BMI feature in heart failure dataset. (Bottom) Calibration plots on heart failure, evaluating at 5 years.

- **RSF**: Random Survival Forest, a black-box tree-based survival model [47].
- **DeepHit**: a black-box model using neural networks [62].
- **DRSA**: a black-box model which uses recurrent neural networks to output survival predictions at multiple evaluation times [86].
- **SATransformer**: a black-box model based on transformers [42].
- **CoxNAM**: glass-box model using Cox loss [121].
- **PseudoNAM**: glass-box model trained using pseudo-values for survival curve [85].

The time-dependent AUC (as defined in scikit-survival [80]) for each dataset and model are shown in Table 2.1. We use time-dependent AUC for model evaluation since all models (except CoxPH and CoxNAM) produce time-dependent predictions. DyS is a top performer on 2 out of 5 datasets, while DNAMite is the best model on two dataset and a top performer on 4 out of 5 datasets. This includes comparisons to black-box models, demonstrating that glass-box survival models like DyS and DNAMite are competitive with state-of-the-art ML models for survival analysis despite being highly interpretable. Further, DNAMite is the best performing glass-box model on 3 out of 5 datasets, showcasing DNAMite as a state-of-the-art glass-box survival model.

### Shape Functions

The top portion of Figure 2.6 shows shape functions for the BMI feature in the heart failure data, using an evaluation time of 5 years. In addition to the glass-box baseline models, we include a version of DNAMite setting $\gamma = 0$ in the embedding module to assess the impact of the kernel smoothing on the learned shape functions, as done in Figure 2.4. The results are similar to the synthetic data results: both DyS and CoxNAM produce shape functions that are too smooth, while DNAMite without smoothing produces a shape function that is too jagged. Meanwhile, DNAMite's shape function is non-monotonic and captures details of the BMI feature without being too noisy.

Table 2.2: Run times (wallclock seconds) for unos and heart_failure.

| model | unos | heart_failure |
|---|---|---|
| CoxPH | 22.895 ± 1.257 | 17.920 ± 2.906 |
| AFT | 12.073 ± 0.282 | 9.378 ± 0.201 |
| mgcv | 981.986 ± 161.866 | 111.261 ± 19.212 |
| SATransformer | 5414.365 ± 602.512 | 2100.597 ± 95.863 |
| DRSA | 1157.576 ± 76.178 | 360.696 ± 26.253 |
| RSF | * | * |
| CoxNAM | 657.999 ± 89.798 | 165.809 ± 31.627 |
| PseudoNAM | * | * |
| DyS | 1517.205 ± 86.914 | 783.335 ± 165.016 |
| DNAMite | 2690.676 ± 103.465 | 819.684 ± 78.972 |

DNAMite's shape function also has the most direct interpretation. In the context of the heart failure dataset, DNAMite's shape function can be interpreted as the contribution of BMI to the predicted probability of developing heart failure in the next 5 years (on the log-odds scale). DyS's shape function, meanwhile, represents the contribution of BMI to the predicted probability of developing heart failure *at* 5 years (not before or after, pre-softmax), which is less useful. CoxNAM's shape function represents the contribution of BMI to the time-independent hazard risk of heart failure, which has no simple clinical interpretation.

#### Calibration

The bottom portion of Figure 2.6 shows calibration plots for the competing methods using the plotting method described in Section 2.2.3, along with the MAE of each plot from the optimal calibration line (shown with dotted line). DNAMite and CoxNAM are both well calibrated, while other models have worse calibration visually and in terms of calibration MAE.

#### Runtimes

Table 2.2 shows the runtimes for all models considered in Table 2.1. All DNAMite runtimes are less than 1 hour, positioning DNAMite as a relatively efficient option even for larger datasets.

## 2.7 Case Study: Heart Failure Prediction

Over 1 million Americans are diagnosed with HF annually, a condition linked to high mortality and projected to cost $70 billion per year by 2030 [11]. Many cases are preventable through early intervention, making accurate prediction crucial [16]. Existing heart failure risk prediction models like PCP-HF [52] and PREVENT-HF [51] were trained using manual feature selection and long-term epidemiological cohorts. We compare these existing risk models to glass-box ML models trained on observational electronic health records (EHR) data.

### 2.7.1 Cohorts

We collect data from two medical systems: Stanford Medical and the Veterans Affairs (VA). In the Stanford cohort, we included adults ($\geq$ 18 years) with $\geq$2 primary care visits and no prior HF, defining the index

Table 2.3: Discrimination of Models in Stanford and VA Cohorts

| Model | Stanford AUC | Stanford C-Index | VA AUC | VA C-Index |
|---|---|---|---|---|
| Base PCP-HF Model | 0.827 | 0.831 | 0.738 | 0.720 |
| Recalibrated PCP-HF Model | 0.836 | 0.840 | 0.722 | 0.706 |
| Base PREVENT-HF Model | 0.835 | 0.837 | **0.764** | **0.740** |
| Recalibrated PREVENT | 0.831 | 0.834 | 0.741 | 0.720 |
| DNAMite-Base | **0.847** | **0.842** | 0.750 | 0.735 |
| DNAMite-Refined | 0.840 | 0.841 | 0.741 | 0.722 |

date as the first primary care visit after January 1, 2015 with at least one year of continuous observation. In the VA cohort, we included adults with $\geq 2$ primary care visits during 2010–2011 and no prior HF, using January 1, 2012 as the index date to allow 10 years of follow-up. Incident HF was defined as $\geq$ 2 outpatient or inpatient encounters with an HF diagnosis. We included ¿2,000 candidate predictors: demographics, diagnoses (from 1-year and indefinite lookbacks), vitals, labs, ECG measurements, and medications. For continuous variables, we included both the most recent value and summary statistics (mean, min, max, SD, frequency). Missing values were not imputed, since DNAMite natively handles missing values.

### 2.7.2 Modeling Approach

We combine the DNAMite architecture with DyS's feature selection functionality to learn feature-sparse additive models. We first train a feature-sparse DNAMite model on the Stanford data, which we call DNAMite-Base. Then, we train a second model called DNAMite-Refined, making the following modifications compared to DNAMite-Base:

1. We model age first and fit all other features on the residual of the age prediction, thereby avoiding selecting features merely due to their correlation with age.
2. We remove non-generalizable or institution-specific features.
3. We combine redundant variables (e.g., multiple eGFR formulas).

We evaluate DNAMite-Base and DNAMite-Refine compared to baseline risk models PCP-HF [52] and PREVENT-HF [51]. For both baseline models, we evaluate the model using the published coefficients as well as refitted version using the Stanford data. We evaluate all models on the Standard data as well as generalized to the VA data.

### 2.7.3 Results

**Predictive Performance** Table 2.3 compares the predictive performance of DNAMite models to the baseline models. On the Stanford data, both DNAMite models outperform all baselines. The base PREVENT-HF model generalizes best to VA data, but both DNAMite models still generalize well and outperform both PCP-HF models. Such generalizability is noteworthy considering that the DNAMite models were built using minimal clinical intervention.

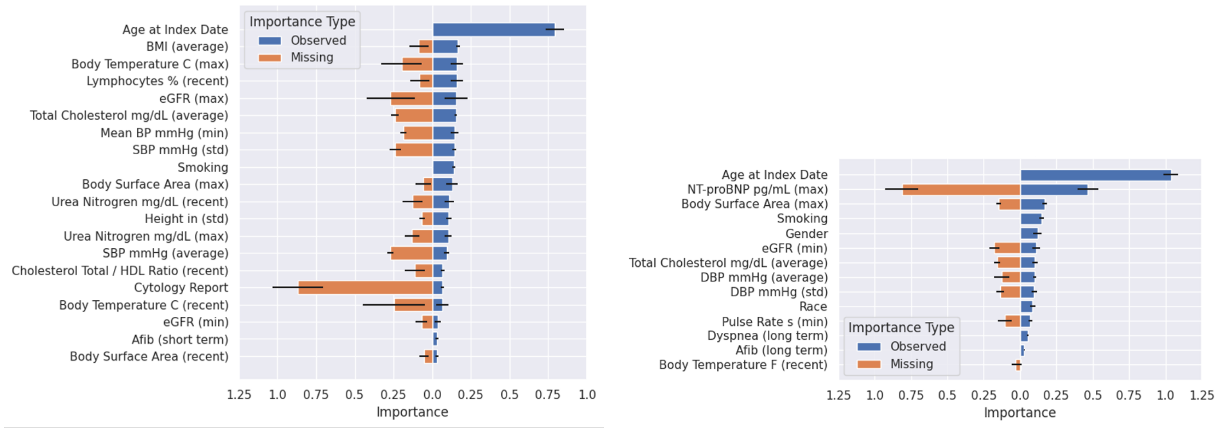


Figure 2.7: Feature importance scores for DNAMite-Base (left) and DNAMite-Refined (right) for 5-year risk prediction.

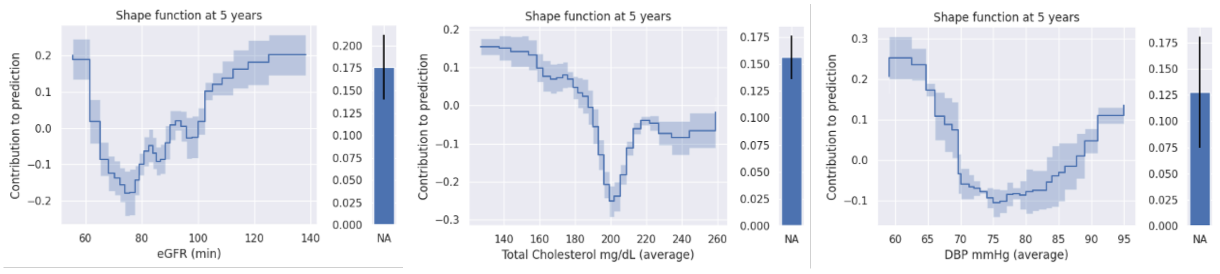


Figure 2.8: Shape functions for 5-year risk for selected features from DNAMite-Refined.

**Features** Figure 2.7 shows the importance score for all features used in DNAMite-Base and DNAMite-Refined. Age is the most important feature in both models. Cytology Report has a very large missing importance score in DNAMite-base, due to correlation with age. Interestingly, there is only moderate overlap between the feature selected by the DNAMite models and the clinical baseline models. Age and smoking are the only risk factors used by all 4 models. 4 risk factors are used by DNAMite-Refined as well as both baseline models: age, smoking, gender, and total cholesterol. Further, DNAMite-Refined and PCP-HF both use race, while DNAMite-Refined and PREVENT both use eGFR. NT-proBNP, the second most important feature in DNAMite-Refined, is not used in PCP-HF or PREVENT.

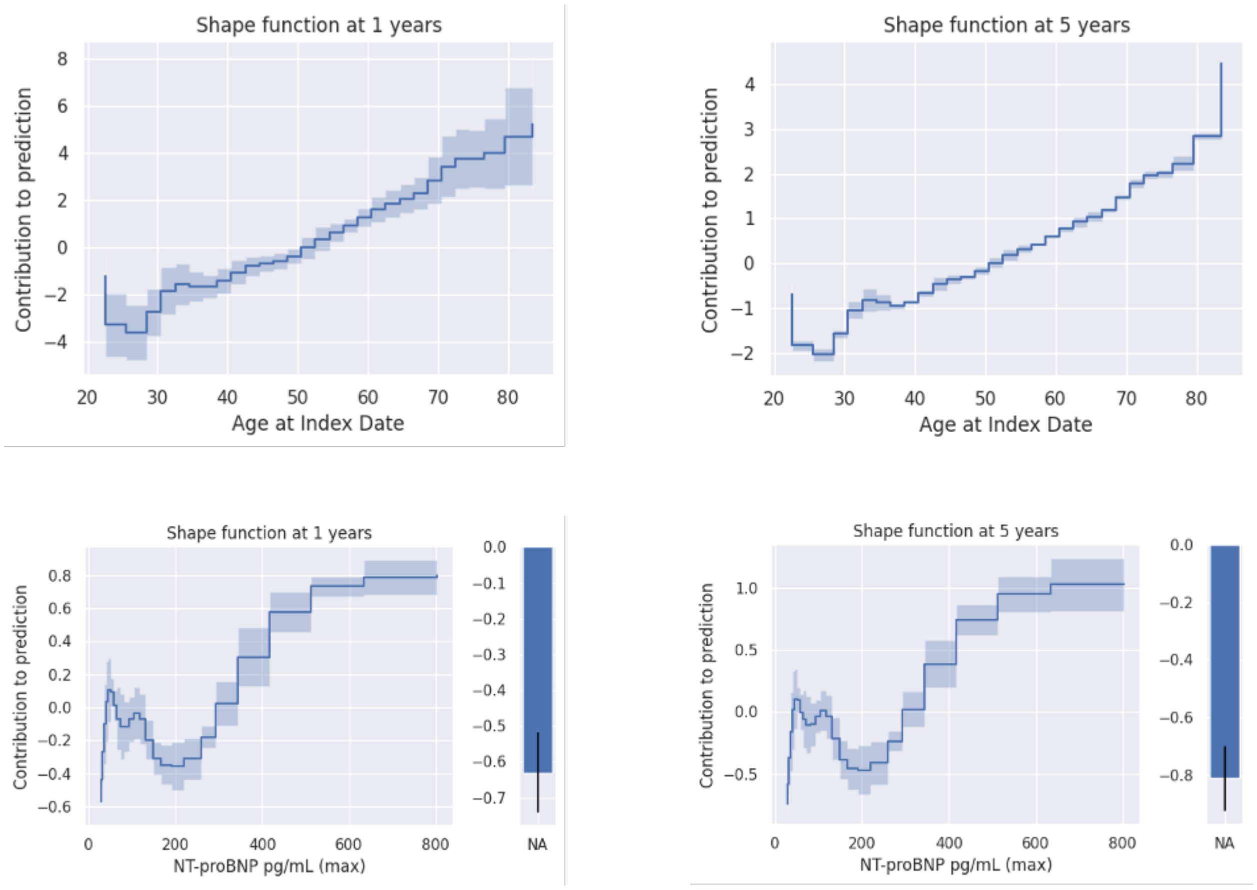
**Shape Functions** Figure 2.8 shows the 5-year risk shape functions for 3 features from DNAMite-Refined. All 3 features show nonlinear trends. For example, DBP (diastolic blood pressure) shows an increased 5-year risk of heart failure for those before 70 mmHg and above 90 mmHg. Shape functions for all features in DNAMite-Base and DNAMite-Refined are shown in Section 2.9.

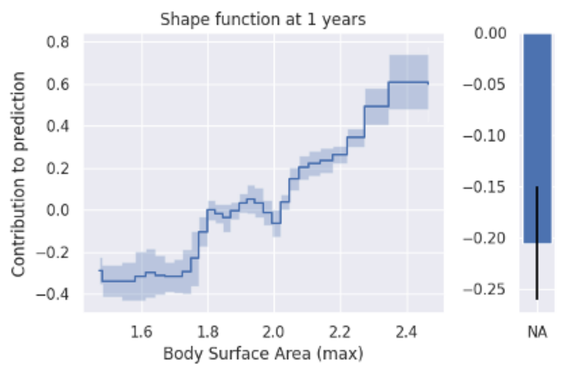
## 2.8 Conclusion

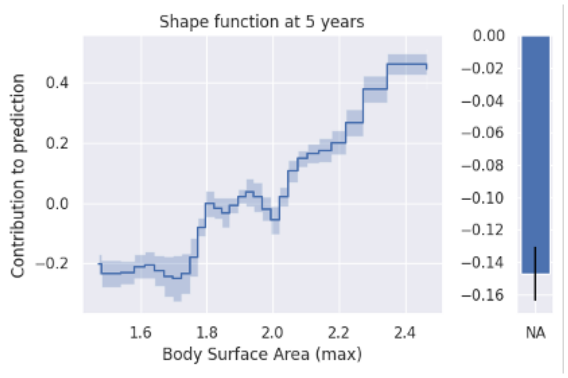
This chapter presents three approaches for building glass-box ML models for survival analysis. The first approach presents a general purpose pipeline for using glass-box binary classification models for survival analysis. DyS and DNAMite, on the other hand, are glass-box ML models specifically designed for survival analysis. Empirical benchmarks show that all three approaches can achieve competitive predictive performance even compared to black-box survival models. Further, the methodology from DyS and DNAMite can be combined to build feature-sparse additive models for survival analysis with discrete shape functions that appropriately balance jaggedness and smoothness. Compared to state-of-the-art clinical models for heart failure risk prediction, this methodology yields models with marginally better predictive performance without the need for clinical intervention.

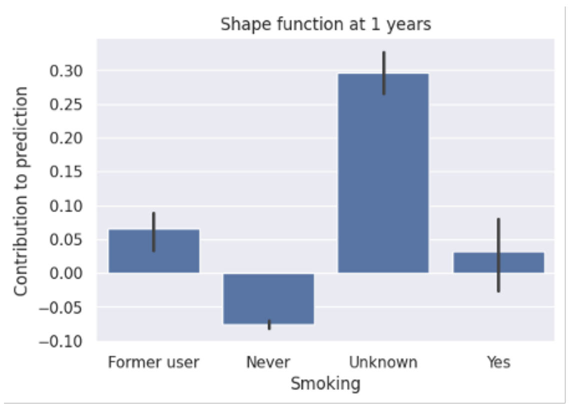
## 2.9 Appendix

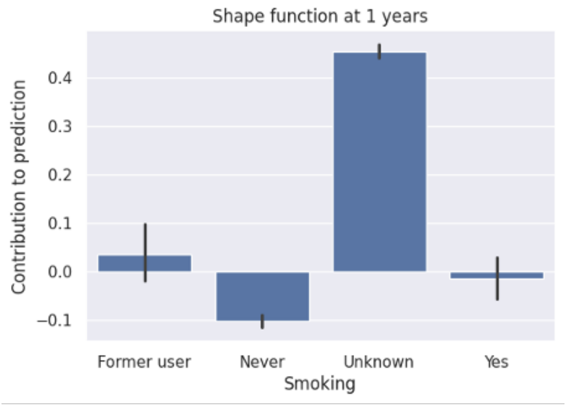
Figure 2.9: Shape functions for DNAMite-Refined at 1 year and 5 years.

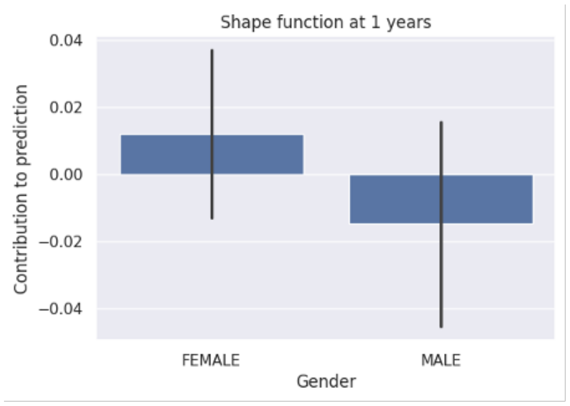


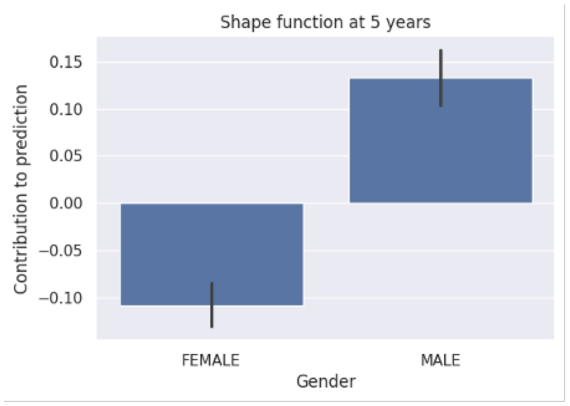

Shape function at 1 years
Contribution to prediction
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
1.6
1.8
2.0
2.2
2.4
Body Surface Area (max)
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
NA

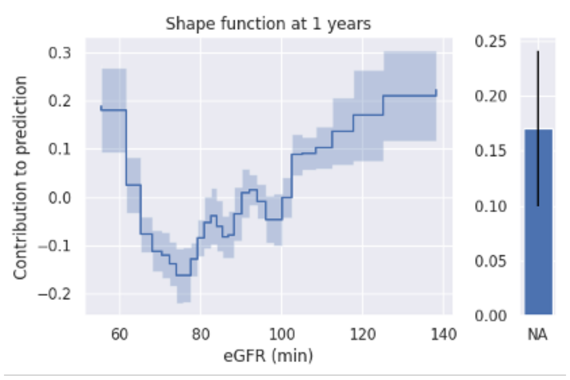

Shape function at 5 years
Contribution to prediction
0.4
0.2
0.0
−0.2
1.6
1.8
2.0
2.2
2.4
Body Surface Area (max)
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
−0.12
−0.14
−0.16
NA

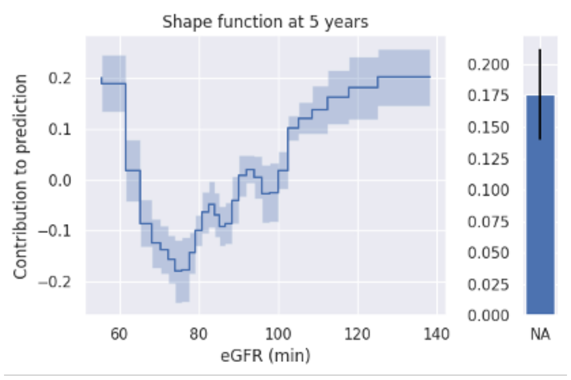

Shape function at 1 years
Contribution to prediction
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
Former user
Never
Unknown
Yes
Smoking

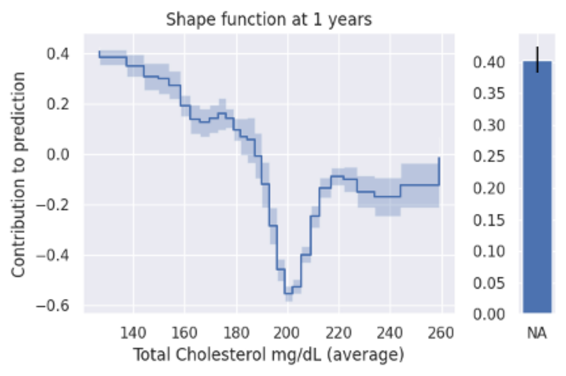

Shape function at 1 years
Contribution to prediction
0.4
0.3
0.2
0.1
0.0
−0.1
Former user
Never
Unknown
Yes
Smoking

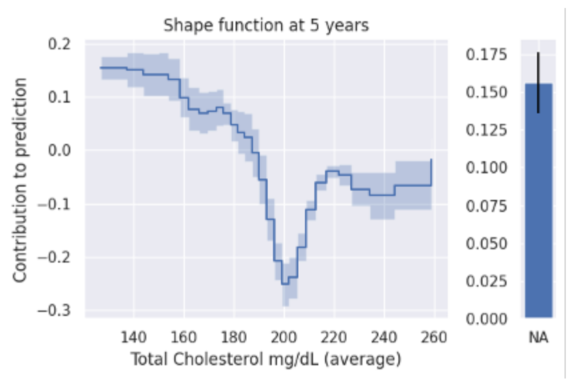

Shape function at 1 years
Contribution to prediction
0.04
0.02
0.00
−0.02
−0.04
FEMALE
MALE
Gender

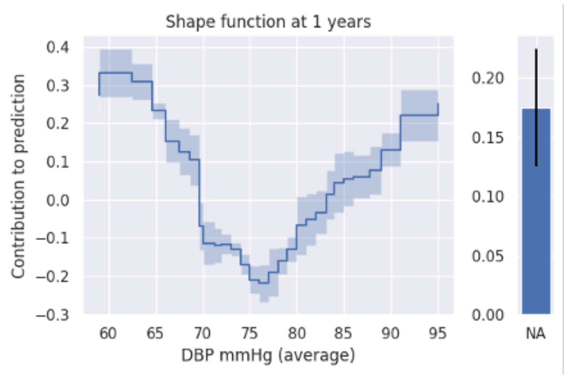

Shape function at 5 years
Contribution to prediction
0.15
0.10
0.05
0.00
−0.05
−0.10
FEMALE
MALE
Gender

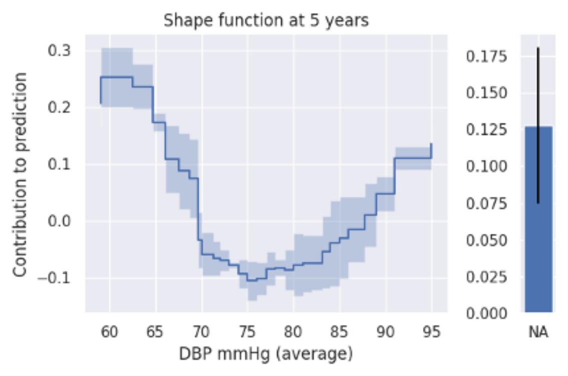

Shape function at 1 years
Contribution to prediction
0.3
0.2
0.1
0.0
−0.1
−0.2
60
80
100
120
140
eGFR (min)
0.25
0.20
0.15
0.10
0.05
0.00
NA

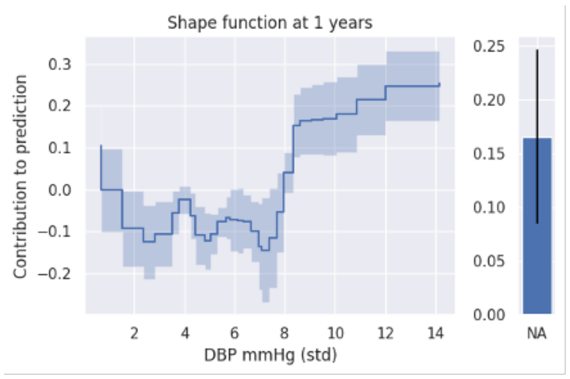

Shape function at 5 years
Contribution to prediction
0.2
0.1
0.0
−0.1
−0.2
60
80
100
120
140
eGFR (min)
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
NA

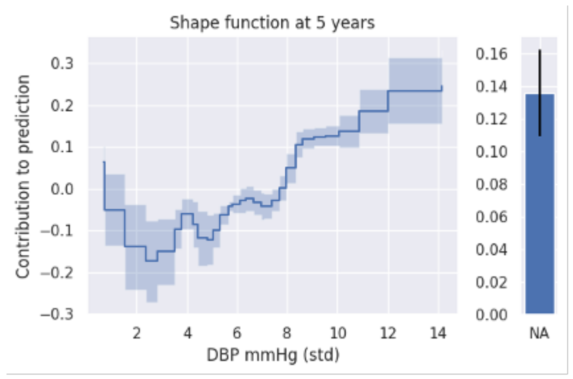

Shape function at 1 years
Contribution to prediction
Total Cholesterol mg/dL (average)
NA

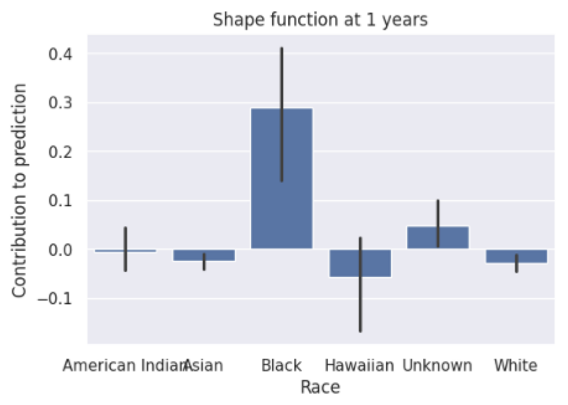

Shape function at 5 years
Contribution to prediction
Total Cholesterol mg/dL (average)
NA

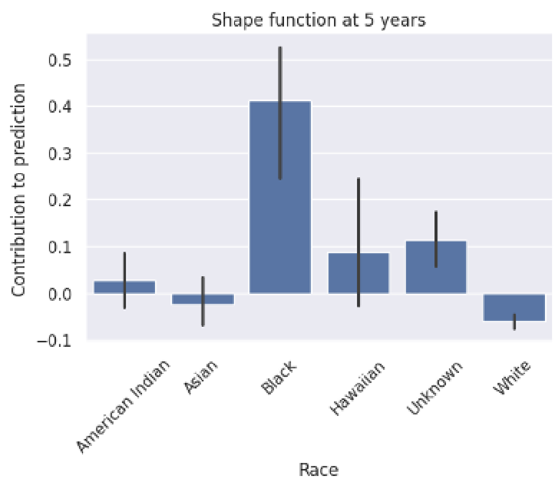

Shape function at 1 years
Contribution to prediction
DBP mmHg (average)
NA

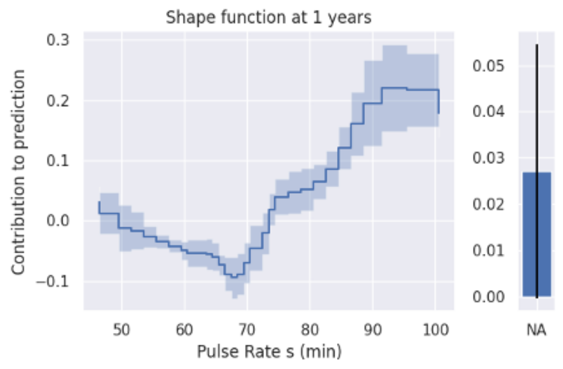

Shape function at 5 years
Contribution to prediction
DBP mmHg (average)
NA

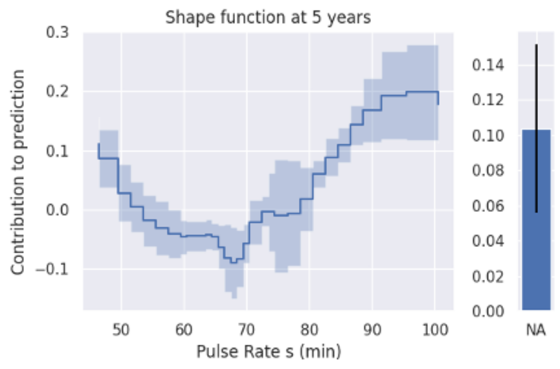

Shape function at 1 years
Contribution to prediction
DBP mmHg (std)
NA

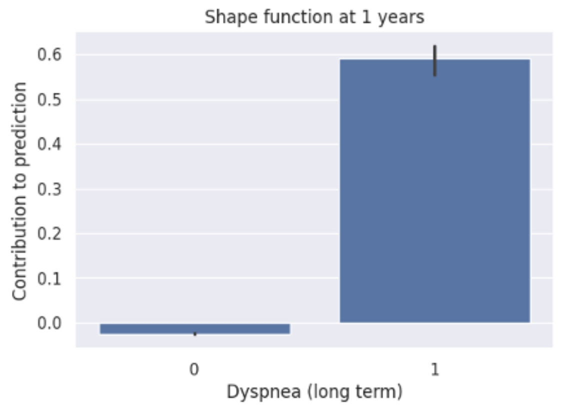

Shape function at 5 years
Contribution to prediction
DBP mmHg (std)
NA

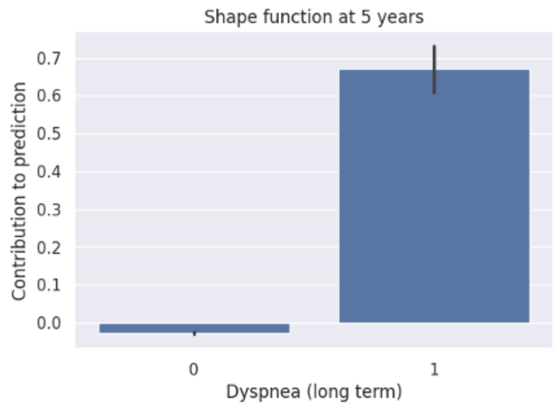

Shape function at 1 years
Contribution to prediction
American Indian
Asian
Black
Hawaiian
Unknown
White
Race


Shape function at 5 years
Contribution to prediction
American Indian
Asian
Black
Hawaiian
Unknown
White
Race


Shape function at 1 years
Contribution to prediction
Pulse Rate s (min)
NA


Shape function at 5 years
Contribution to prediction
Pulse Rate s (min)
NA


Shape function at 1 years
Contribution to prediction
Dyspnea (long term)


Shape function at 5 years
Contribution to prediction
Dyspnea (long term)

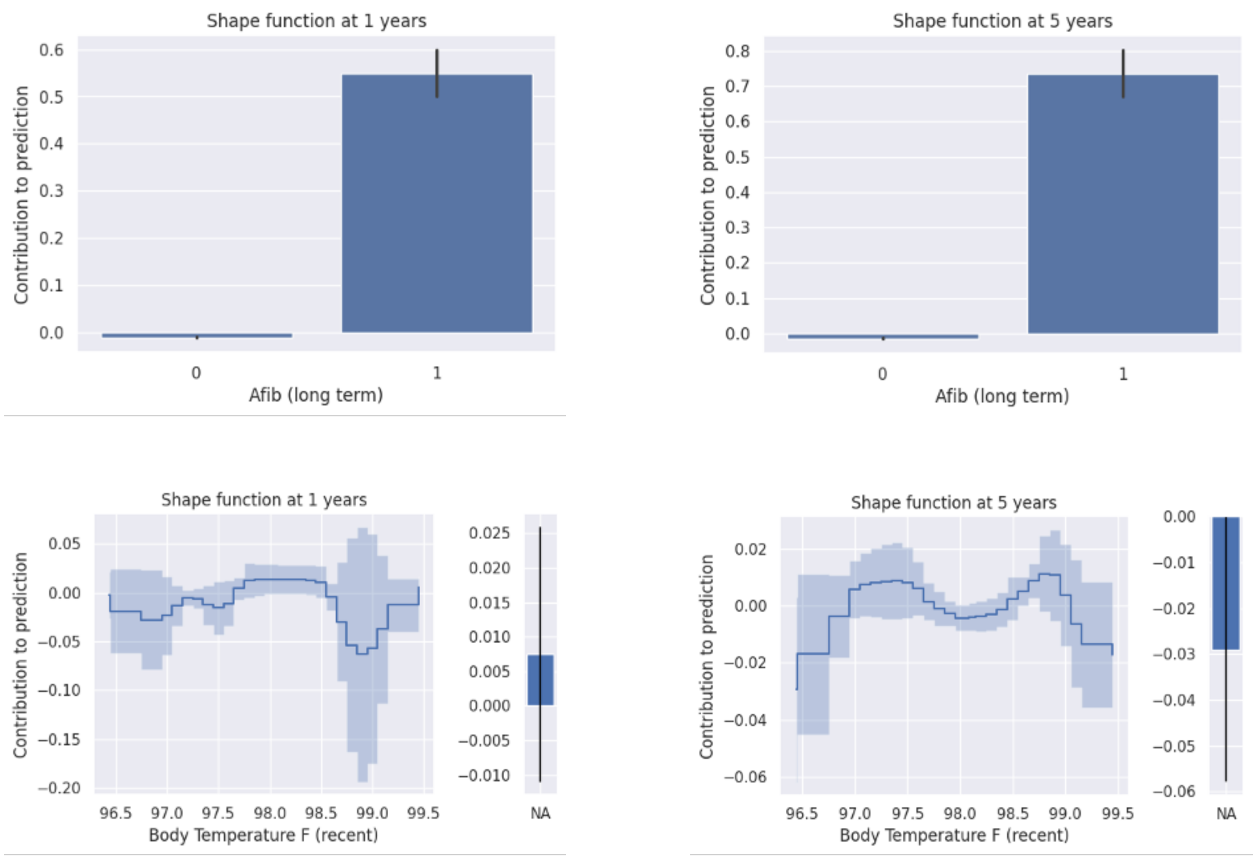
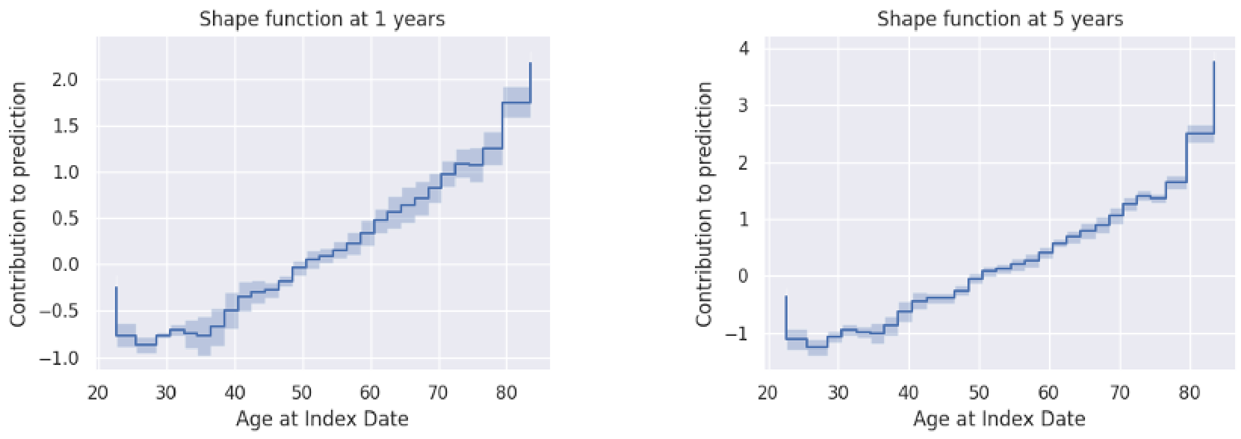
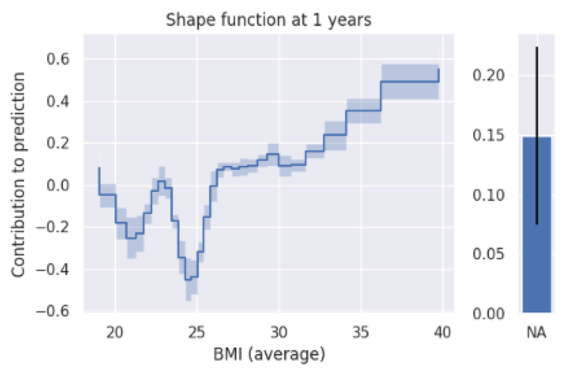
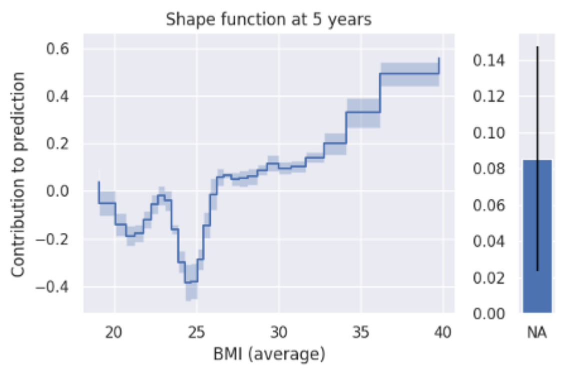
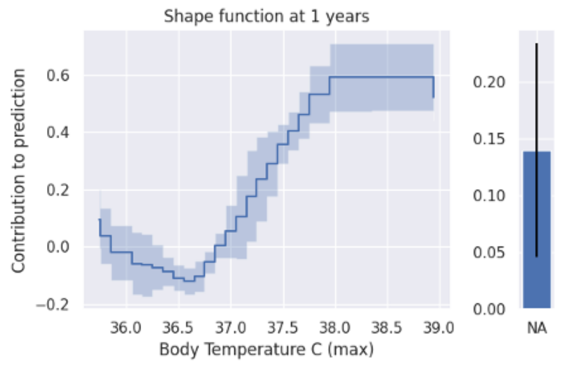


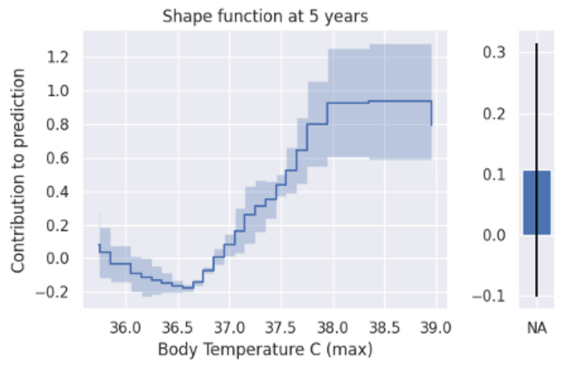
Figure 2.10: Shape functions for DNAMite-Base at 1 year and 5 years.

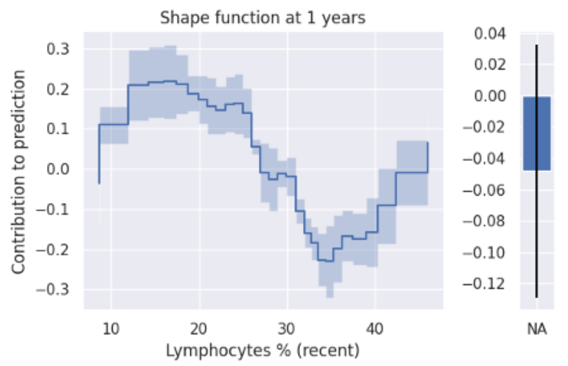


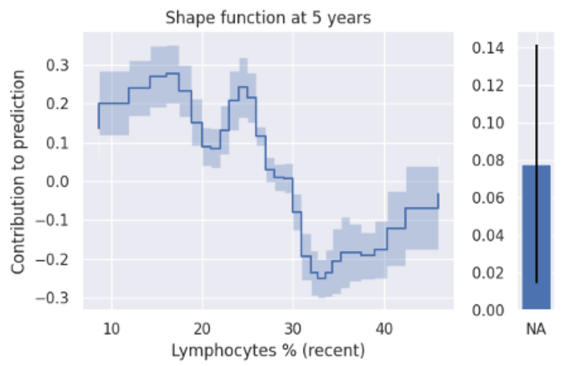

Shape function at 1 years
Contribution to prediction
BMI (average)
NA

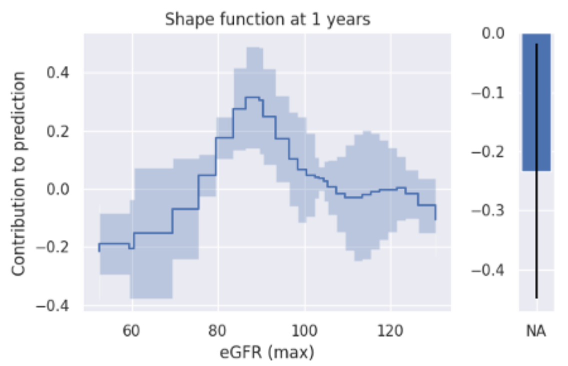

Shape function at 5 years
Contribution to prediction
BMI (average)
NA

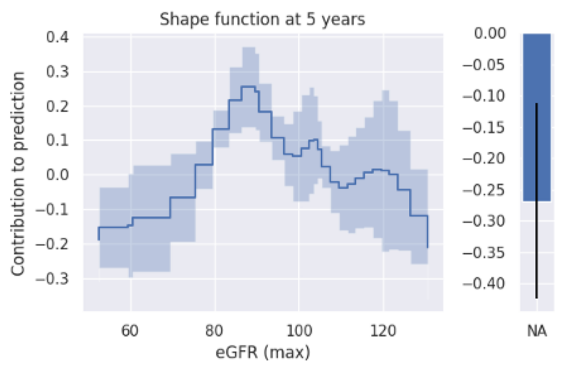

Shape function at 1 years
Contribution to prediction
Body Temperature C (max)
NA

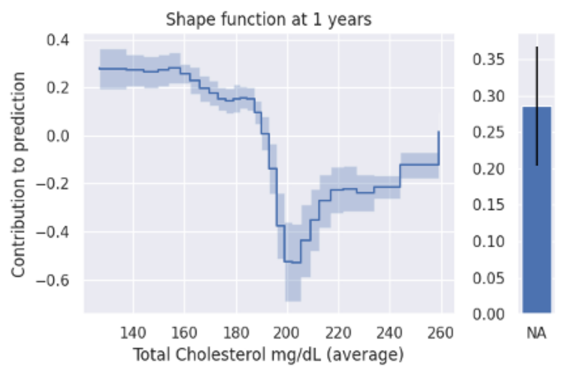

Shape function at 5 years
Contribution to prediction
Body Temperature C (max)
NA

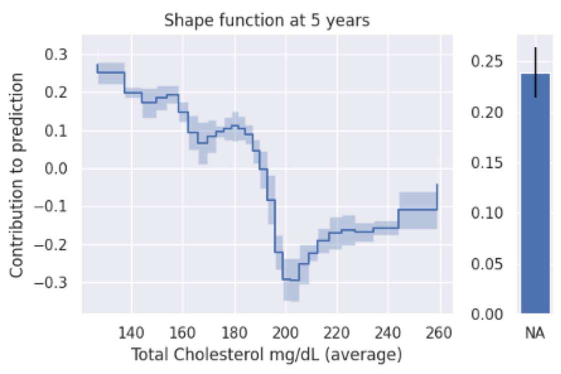

Shape function at 1 years
Contribution to prediction
Lymphocytes % (recent)
NA

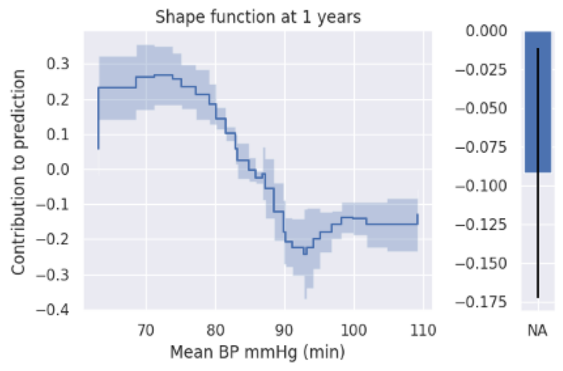

Shape function at 5 years
Contribution to prediction
Lymphocytes % (recent)
NA

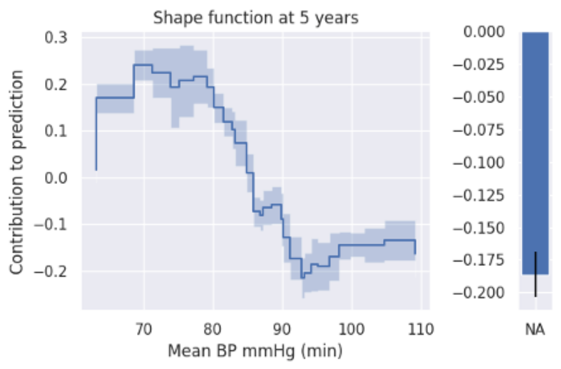

Shape function at 1 years
Contribution to prediction
eGFR (max)
NA

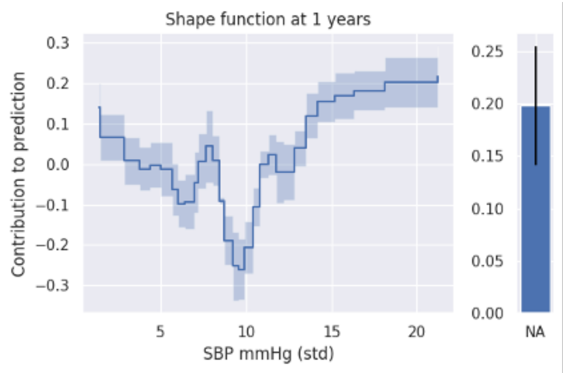

Shape function at 5 years
Contribution to prediction
eGFR (max)
NA

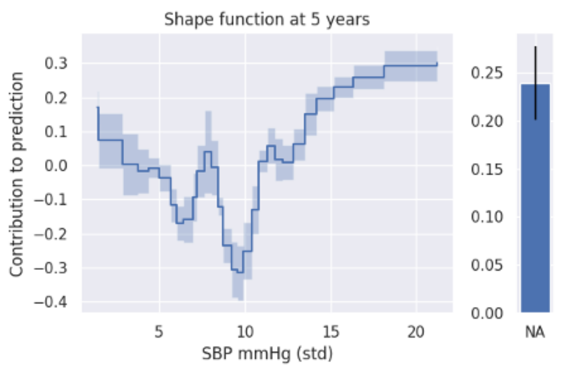

Shape function at 1 years
Contribution to prediction
0.4
0.2
0.0
−0.2
−0.4
−0.6
140
160
180
200
220
240
260
Total Cholesterol mg/dL (average)
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
NA

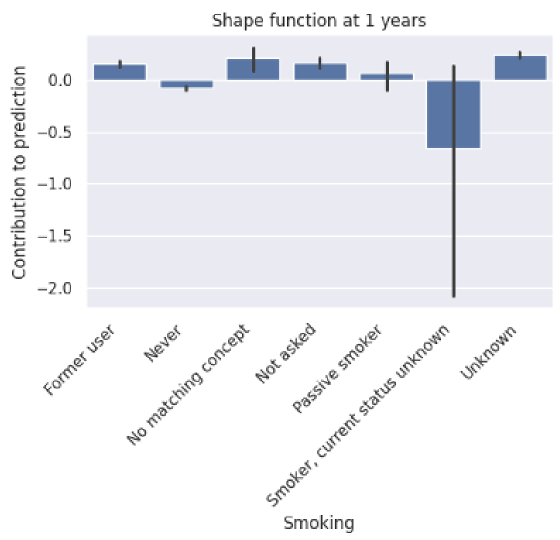

Shape function at 5 years
Contribution to prediction
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
140
160
180
200
220
240
260
Total Cholesterol mg/dL (average)
0.25
0.20
0.15
0.10
0.05
0.00
NA

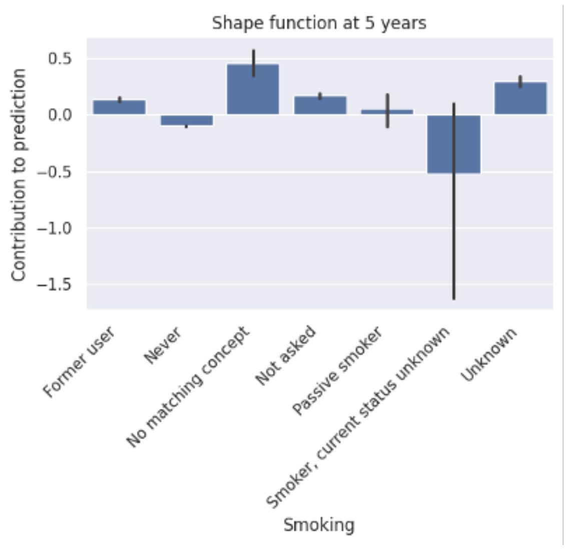

Shape function at 1 years
Contribution to prediction
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
70
80
90
100
110
Mean BP mmHg (min)
0.000
−0.025
−0.050
−0.075
−0.100
−0.125
−0.150
−0.175
NA

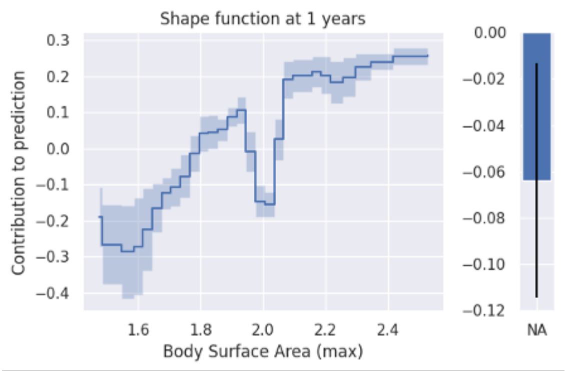

Shape function at 5 years
Contribution to prediction
0.3
0.2
0.1
0.0
−0.1
−0.2
70
80
90
100
110
Mean BP mmHg (min)
0.000
−0.025
−0.050
−0.075
−0.100
−0.125
−0.150
−0.175
−0.200
NA

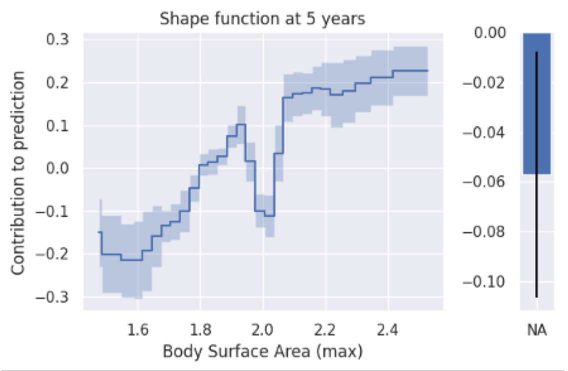

Shape function at 1 years
Contribution to prediction
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
5
10
15
20
SBP mmHg (std)
0.25
0.20
0.15
0.10
0.05
0.00
NA

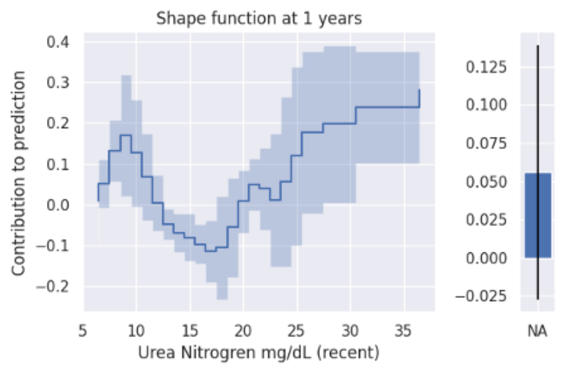

Shape function at 5 years
Contribution to prediction
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
5
10
15
20
SBP mmHg (std)
0.25
0.20
0.15
0.10
0.05
0.00
NA

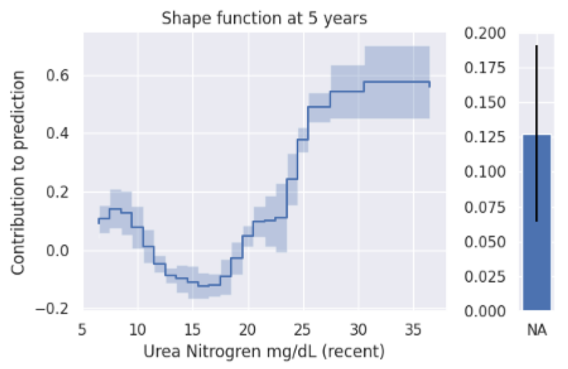

Shape function at 1 years
Contribution to prediction
Former user
Never
No matching concept
Not asked
Passive smoker
Smoker, current status unknown
Unknown
Smoking

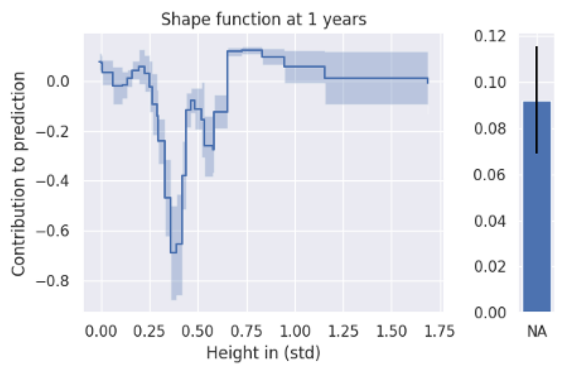

Shape function at 5 years
Contribution to prediction
Former user
Never
No matching concept
Not asked
Passive smoker
Smoker, current status unknown
Unknown
Smoking

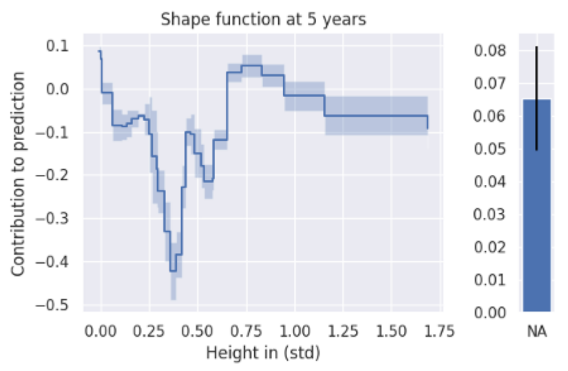

Shape function at 1 years
Contribution to prediction
Body Surface Area (max)
NA

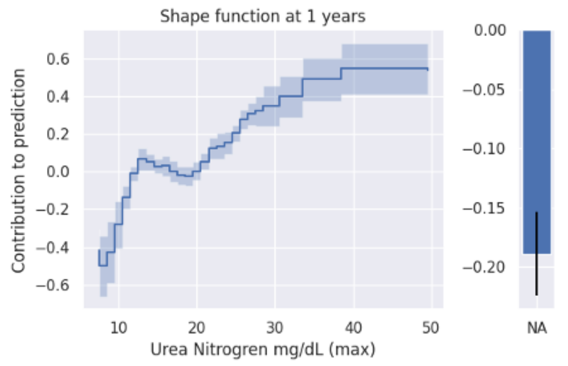

Shape function at 5 years
Contribution to prediction
Body Surface Area (max)
NA

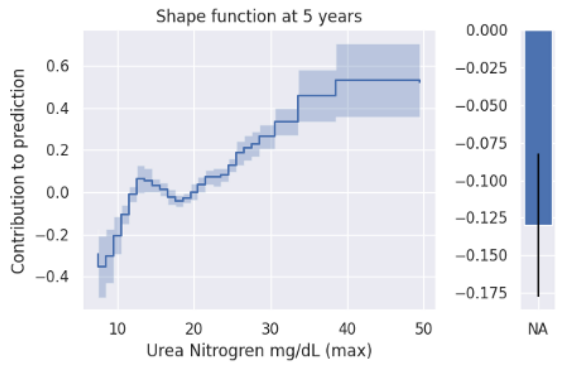

Shape function at 1 years
Contribution to prediction
Urea Nitrogren mg/dL (recent)
NA

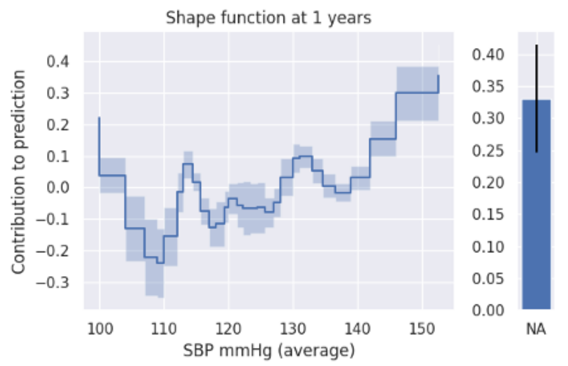

Shape function at 5 years
Contribution to prediction
Urea Nitrogren mg/dL (recent)
NA

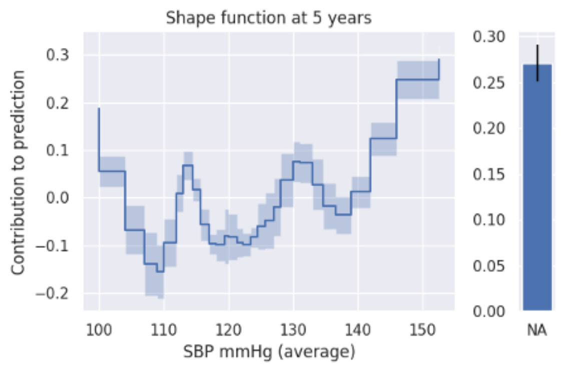

Shape function at 1 years
Contribution to prediction
Height in (std)
NA

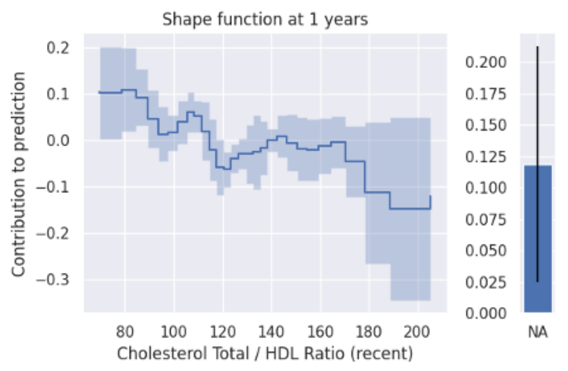

Shape function at 5 years
Contribution to prediction
Height in (std)
NA

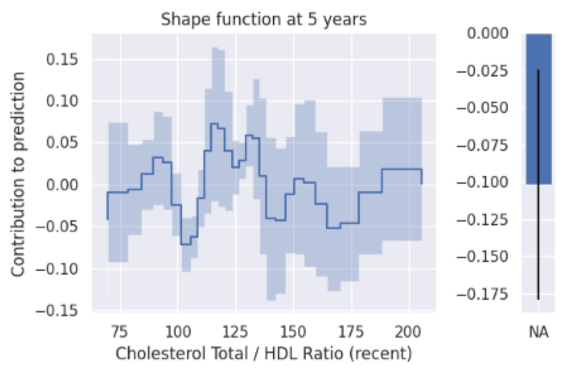

Shape function at 1 years
Contribution to prediction
Urea Nitrogren mg/dL (max)
NA

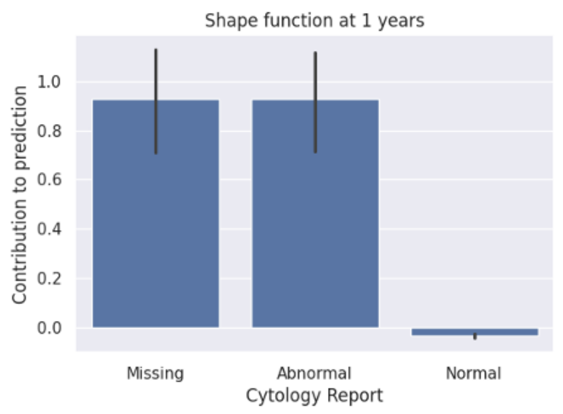

Shape function at 5 years
Contribution to prediction
Urea Nitrogren mg/dL (max)
NA

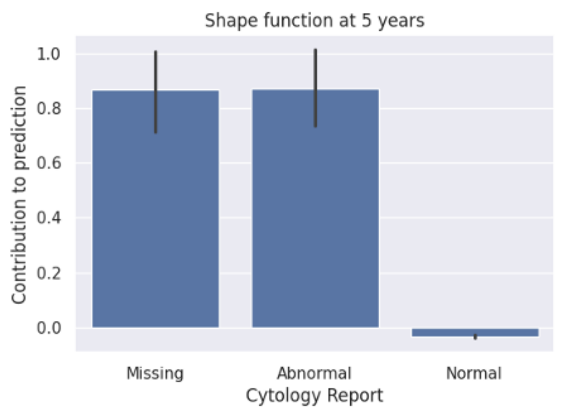

Shape function at 1 years
Contribution to prediction
SBP mmHg (average)
NA

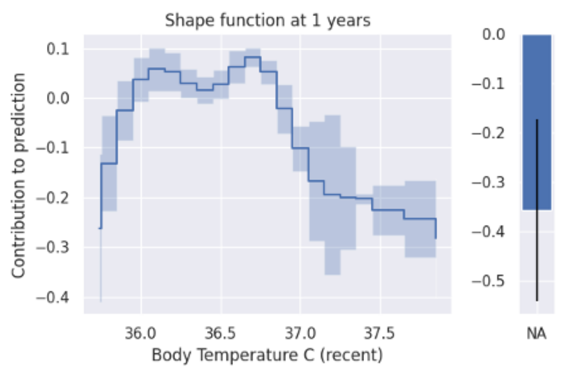

Shape function at 5 years
Contribution to prediction
SBP mmHg (average)
NA

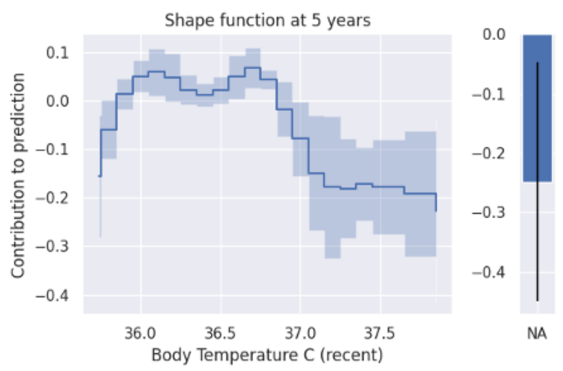

Shape function at 1 years
Contribution to prediction
Cholesterol Total / HDL Ratio (recent)
NA

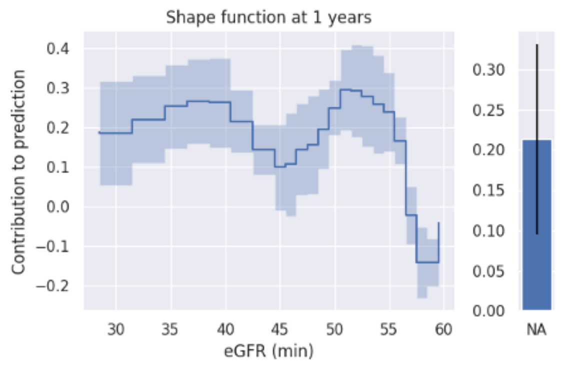

Shape function at 5 years
Contribution to prediction
Cholesterol Total / HDL Ratio (recent)
NA

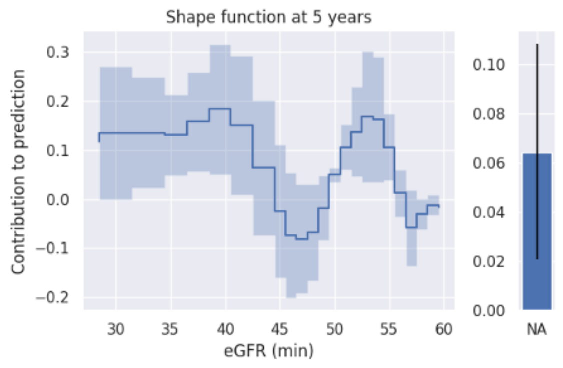

Shape function at 1 years
Contribution to prediction
Missing
Abnormal
Normal
Cytology Report

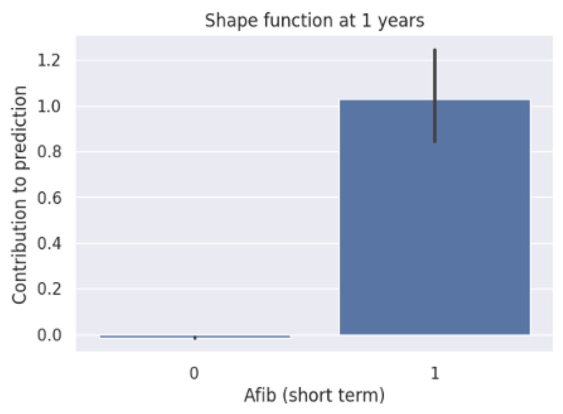

Shape function at 5 years
Contribution to prediction
Missing
Abnormal
Normal
Cytology Report

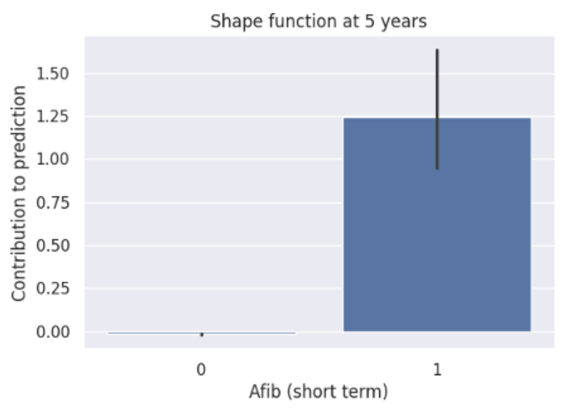

Shape function at 1 years
Contribution to prediction
Body Temperature C (recent)
NA

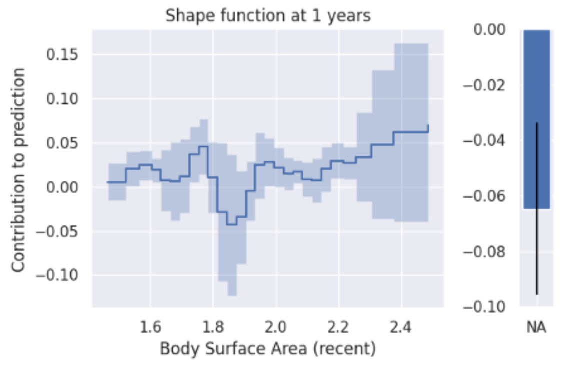

Shape function at 5 years
Contribution to prediction
Body Temperature C (recent)
NA

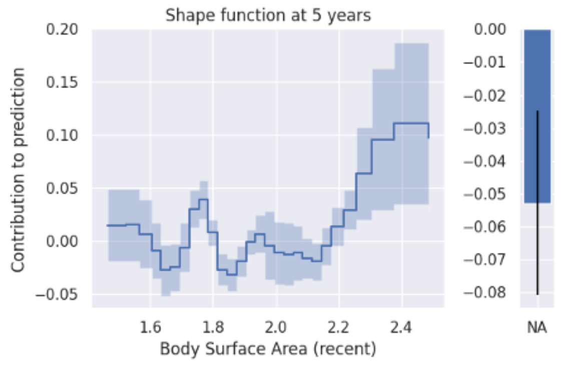

Shape function at 1 years
Contribution to prediction
eGFR (min)
NA


Shape function at 5 years
Contribution to prediction
eGFR (min)
NA


Shape function at 1 years
Contribution to prediction
Afib (short term)
0
1


Shape function at 5 years
Contribution to prediction
Afib (short term)
0
1


Shape function at 1 years
Contribution to prediction
Body Surface Area (recent)
NA


Shape function at 5 years
Contribution to prediction
Body Surface Area (recent)
NA

# Chapter 3

# dnamite: a Python Package for Neural Additive Models

Neural additive models (NAMs) provide many benefits compared to traditional additives, as discussed in Chapter 2. While standard additive models have support through mature packages in R [119] and Python [93], there is currently no package in R or Python for NAMs that supports both survival analysis and feature selection. dnamite [114], a Python package for neural additive modeling, fills these gaps. dnamite implements NAMs via the DNAMite architecture [110], which was originally proposed for survival analysis but can be easily adapted to support regression and classification. Additionally, dnamite supports feature selection for regression, classification, and survival analysis using the learnable gates method in DyS. Most importantly, dnamite is fully compatible with scikit-learn, enabling seamless training and analysis through simple function calls and ensuring smooth integration into the scikit-learn ecosystem. dnamite is open-source, publicly available on github, registered for installation via pip, and fully documented at `https://dnamite.readthedocs.io`.

## 3.1 Design Principles

dnamite provides users with an easy-to-use interface for training and interacting with NAMs. dnamite is designed in the style of scikit-learn, allowing users to train, extract information, and make predictions from a model with simple function calls directly from pandas dataframes. This design contrasts with many neural network implementations that expect users to prepare specific data loaders and write custom training loops. Although dnamite is written in PyTorch (thus scaling seamlessly on multi-core CPU or GPU hardware), users of dnamite do not need to write any PyTorch code.

Figure 3.1 shows the class structure used in dnamite. The base class `BaseDNAMiteModel` serves as the parent class for all dnamite models, implementing common functionality such as high-level functions for model fitting, predicting, and making explanation plots. For each of the three supervised learning tasks that dnamite supports (regression, binary classification, survival analysis), dnamite implements a child class of `BaseDNAMiteModel` with task-specific functionality such as training loops. When a `BaseDNAMiteModel`

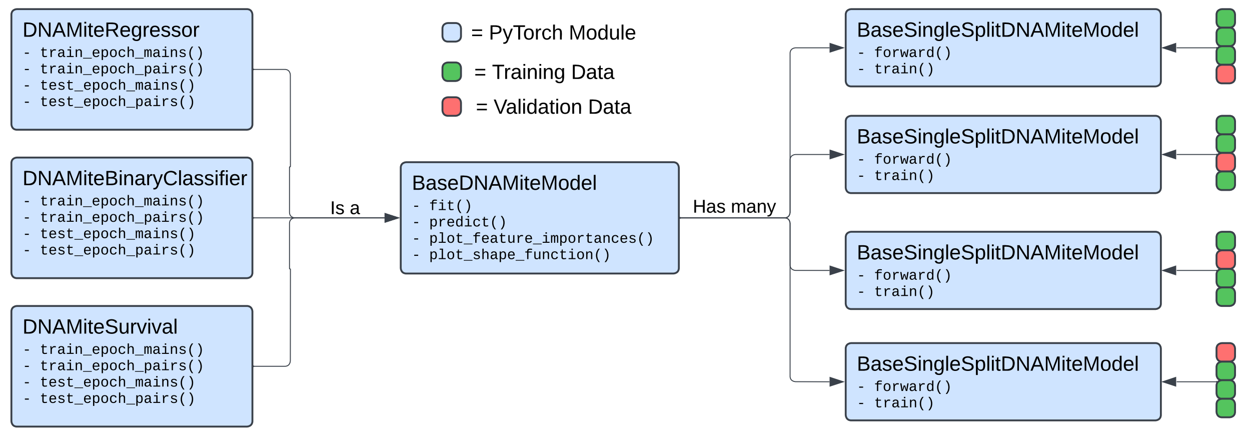


Figure 3.1: Class structure for dnamite models.

is trained, $k$ instances of `BaseSingleSplitDNAMiteModel` are trained, each on a different train/validation split. Each `BaseSingleSplitDNAMiteModel` implements the DNAMite architecture from Figure 2.3. DyS-style feature gates as in Figure 2.2 are additionally added to enable feature selection. Predictions, feature importances, and shape functions for a `BaseDNAMiteModel` are all computed by averaging the results across the single split models. This cross-validation approach allows dnamite to compute errors for feature importance and shape function plots, which is critical for assessing the reliability of these plots.

## 3.2 Related Packages

Several existing packages can be used to train additive models, as summarized in Table 3.1. Most existing packages focus on fitting splines for each shape function, including many R packages [36, 119, 65, 91] and a Python package [93]. A notable exception is the Python package Interpretml [75], which uses boosted decision trees. Additionally, some R packages extend additive models to more advanced applications. For example, mboost [39] supports both trees and splines and employs component-wise gradient boosting to enable large-scale training and feature sparsity. gamlss [98] trains additive models that output full distributions rather than just the mean response. Several packages build on this paradigm: gamboostLSS [40] combines the benefits of mboost and gamlss, while bamlss [105] trains fully Bayesian distributional additive models. dnamite instead uses neural networks for shape functions, leveraging their flexibility to naturally handle missing values, perform feature selection, and conduct fully nonparametric survival analysis.

In comparison, very few packages exist for training NAMs. PiML [100] implements several NAM architectures, but only one (GAMI-Net [123]) that supports feature-sparsity and none that support survival analysis. The R package neuralGAM [77] also has a Python version but with no support for feature selection or survival analysis. dnamite, meanwhile, implements a NAM architecture that has been shown to fit shape functions more accurately than standard NAM architectures [110].

Table 3.1: Comparison of additive model packages.

| Package | Language | Models | Feature Selection | Survival Analysis | Missing Values |
|---|---|---|---|---|---|
| gam | R | Splines | ✗ | Cox | ✗ |
| mgcv | R | Splines | ✗ | Cox/AFT | ✗ |
| mboost | R | Splines/Trees | Boosting | Cox/AFT | ✗ |
| gamlss | R | Splines | ✗ | Parametric | ✗ |
| gamboostLSS | R | Splines/Trees | Boosting | Parametric | ✗ |
| bamlss | R | Splines | Lasso (linear) | Parametric | ✗ |
| cgam | R | Splines | ✗ | ✗ | ✗ |
| spikeSlabGAM | R | Splines | Spike-and-Slab | ✗ | ✗ |
| neuralGAM | R | Neural Nets | ✗ | ✗ | ✗ |
| pyGAM | Python | Splines | ✗ | ✗ | ✗ |
| interpretml | Python | Trees | Interactions only | ✗ | ✓ |
| PiML | Python | Trees/Neural Nets | Post-Hoc | ✗ | ✗ |
| dnamite | Python | Neural Nets | Learnable Gates | Nonparametric | ✓ |

## 3.3 Usage

In this section, we demonstrate how to use dnamite through an empirical example. Given the importance of interpretability and feature selection in healthcare applications, we focus our experiments on the MIMIC III clinical database [48], an electronic health records database from the Beth Israel Deaconess Medical Center. Since MIMIC III is open source and anonymized, it is a natural choice for reproducible healthcare data science experiments. For more usage information, see dnamite's official documentation[1] and source code[2].

### 3.3.1 Data Preparation

Preparing data for dnamite requires very little preprocessing. All dnamite models ingest features as a pandas dataframe and labels as a pandas series or numpy array. Features can be categorical, continuous, or even missing, simplifying data processing: categorical features need not be encoded, continuous features need not be standardized, and missing values need not be imputed as dnamite naturally handles all of these data types.

To prepare the raw MIMIC III data for our usage demonstration, we identify a cohort of 36,639 patients who are at least 18 years old and have a minimum of 48 hours between their first admission and their last discharge or death. Our focus is on mortality prediction, with two subtasks: in-hospital mortality prediction as a binary classification task, and time-to-death prediction from the index time as a survival analysis problem.

We generate two mortality prediction datasets from the raw MIMIC III data. The first is a comprehensive dataset with 757 features extracted from labevents and chartevents. The second is a benchmark dataset consisting of one feature for each of the 17 clinical variables used in the MIMIC III benchmark [35], see Table 3.2. Most features align with well-known medical concepts. One lesser-known concept is the Glasgow Coma Scale (GCS), which assesses consciousness through eye, motor, and verbal responses. Lower values in these categories indicate reduced responsiveness.

We can generate these datasets from the raw MIMIC III data using dnamite's `fetch_mimic` function.

[1]`https://dnamite.readthedocs.io/`
[2]`https://github.com/udellgroup/dnamite`

| Feature | Type | Missing Rate | Example Values |
|---|---|---|---|
| Age | Continuous | 0.000 | 74.0, 23.0, 20.0 |
| Gender | Categorical | 0.000 | ‘F’, ‘F’, ‘M’ |
| Ethnicity | Categorical | 0.000 | ‘WHITE’, ‘BLACK’, ‘UNKNOWN’ |
| Diastolic BP (mmHg) | Continuous | 0.173 | 55.107, 65.483, 56.043 |
| Inspired oxygen (%) | Continuous | 0.814 | 52.0, 50.0, 70.0 |
| GCS Total | Continuous | 0.522 | 11.889, 7.143, 14.526 |
| GCS Eye Opening | Continuous | 0.175 | 3.667, 2.143, 3.842 |
| GCS Motor Response | Continuous | 0.175 | 5.889, 3.143, 6.000 |
| GCS Verbal Response | Continuous | 0.175 | 2.333, 1.857, 4.684 |
| Glucose (mg/dL) | Continuous | 0.177 | 208.111, 154.611, 108.5 |
| Heart Rate (bpm) | Continuous | 0.173 | 94.677, 135.775, 69.875 |
| Height (in) | Continuous | 0.847 | 72.0, 60.0, 64.0 |
| Mean Blood Pressure (mmHg) | Continuous | 0.173 | 72.071, 95.246, 78.159 |
| Oxygen Saturation (%) | Continuous | 0.172 | 97.906, 91.637, 99.25 |
| Respiratory Rate (bpm) | Continuous | 0.173 | 13.565, 21.226, 17.125 |
| Systolic Blood Pressure (mmHg) | Continuous | 0.173 | 110.262, 120.846, 122.391 |
| Temperature (F) | Continuous | 0.186 | 98.7, 99.51, 98.12 |
| Weight (kg) | Continuous | 0.700 | 51.2, 87.0, 78.3 |
| pH | Continuous | 0.404 | 7.282, 7.279, 7.46 |
| Capillary refill rate | Categorical | 0.774 | ‘Brisk”, ‘Brisk’, ‘Delayed’ |

Table 3.2: Features generated from MIMIC III for the benchmark dataset, following [35].

```python
from dnamite.datasets import fetch_mimic
data_path = "/home/mvanness/mimic/mimic_raw_data/"
cohort_data, benchmark_data = fetch_mimic(
    data_path=data_path,
    return_benchmark_data=True
)
```

See the package documentation for complete details on data generation. The input file path should be adjusted to point to a directory with the raw MIMIC III csv files, which may be obtained from physionet[3].

## 3.3.2 Basic Usage

We start by importing standard packages and defining a few useful variables.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme()
from sklearn.model_selection import train_test_split
import torch
import os
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

[3]https://physionet.org/content/mimiciii/1.4/

```
seed = 10
```

The most basic use of dnamite is as an additive model for regression or classification. This paper demonstrates binary classification, but regression is very similar. The following code prepares the benchmark in-hospital mortality binary classification dataset.

```
X_benchmark = benchmark_data.drop(
    columns=["SUBJECT_ID", "HOSPITAL_EXPIRE_FLAG"]
)
y_benchmark = benchmark_data["HOSPITAL_EXPIRE_FLAG"]
X_train_benchmark, X_test_benchmark, \
y_train_benchmark, y_test_benchmark = train_test_split(
    X_benchmark,
    y_benchmark,
    test_size=0.2,
    random_state=seed,
)
```

Fitting a binary classification dnamite model follows the standard scikit-learn style API, using the `DNAMiteBinaryClassifier` class. After initialization, we train a dnamite model using the `fit` function.

```
from dnamite.models import DNAMiteBinaryClassifier
model = DNAMiteBinaryClassifier(
   random_state=seed,
   device=device,
   num_pairs=5
)
model.fit(X_train_benchmark, y_train_benchmark)
```

`DNAMiteBinaryClassifier` has several parameters that can be set at initialization, the full list of which is available in the package documentation. Three optional parameters are specified:

- `random_state`: a random seed for controlling reproducibility.
- `device`: a PyTorch device or string used to initialize a PyTorch device. The default is “cpu”. Passing a cuda device or string name triggers GPU training, which is more efficient.
- `num_pairs`: the number of pairwise interactions to use. To select exactly $k$ interactions, dnamite keeps the $k$ interactions with the largest gates $s(\mu_{j,k})$. The default is zero, which excludes interactions from the model.

When we call the `fit` function, several steps happen under the hood.

1. **Identifying Data Types**: dnamite first automatically determines the data type of each feature in the input training data. The three possible data types are continuous, binary, and categorical.

2. **Defining Bins**: dnamite defines feature bins for each feature in order to prepare for discretization and embedding. Two optional parameters can be passed to a dnamite model that affect the binning of continuous features. First, `max_bins` controls the maximum number of bins a feature can have. The default is 32. Second, `min_samples_per_bin` sets a lower bound on the number of samples from the training dataset that must belong to each bin. The default is the minimum of 50 and 1% of the training data samples. Since each bin is assigned an embedding, this safeguard ensures that each embedding has enough data to properly train.
3. **Fitting Single Split Models**: After defining data types and feature bins, dnamite starts fitting single split DNAMite models, each to a different train/validation split (see Figure 3.1). The optional parameter `n_val_splits`, which defaults to 5, controls how many data splits are used. For a given split, dnamite uses the defined bins to transform the training and validation data so that all values are replaced by their corresponding bin index. This transformed data is fed to PyTorch data loaders to prepare the data for training. If `num_pairs` is set to a value greater than 0, then pairs are selected using the first split and remain the same for all splits.

We can save and load `dnamite` models using the standard PyTorch saving and loading functions.

```
torch.save(model, "mimic_benchmark_model.pth")
model = torch.load("mimic_benchmark_model.pth")
```

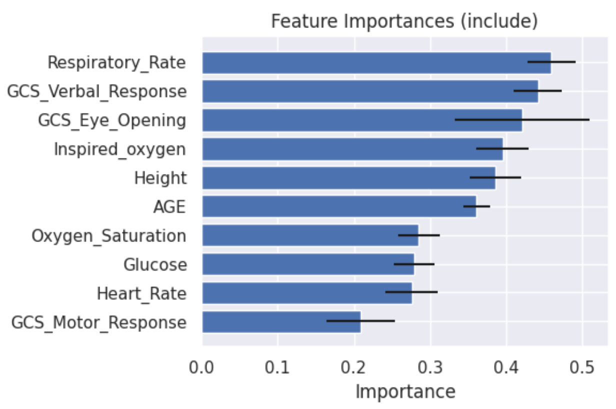
Next, we use the `predict_proba` function to get probabilistic predictions from the model, which are needed to compute the AUC score.

```
from sklearn.metrics import roc_auc_score
preds = model.predict_proba(X_test_benchmark)
dnamite_auc = np.round(roc_auc_score(y_test_benchmark, preds), 4)
dnamite_auc
```

```
0.8521
```

After training a dnamite model, we can visualize feature importances and shape functions with very little additional computation. We first use the `plot_feature_importances` function to plot the feature importance scores of the top features in descending order.

```
model.plot_feature_importances()
```



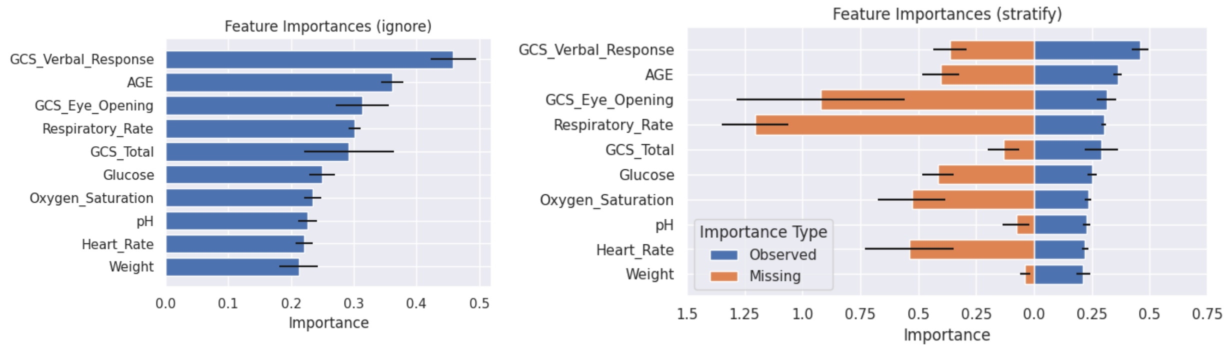
Each bar represents the mean importance across validation splits, with error bars representing confidence intervals. The optional parameter `n_features` can be used to change how many features are plotted. If the model uses pairwise interactions, these are also each assigned a score and eligible for inclusion in the plot. For example, if feature "A" and feature "B" had an important interaction, it would be displayed as "A || B".

A second optional parameter `missing_bin` changes how the missing bin is used in the feature importance calculation. The following are the options for setting this parameter.

- `"include"`: the default, which treats the missing bin the same as all other bins. This option is acceptable for most use cases but can result in misleadingly high importances if a feature's missing bin has a high absolute score due to informative missingness.
- `"ignore"`: disregard the missing bin, i.e. average the absolute value of the shape function only for samples where the feature is not missing. This option is useful if a user does not want missing values to influence feature importances, or if diagnosing and analyzing informative missingness is not important.
- `"stratify"`: compute separate feature importance scores for missing and observed. The observed score is the same as the score when using "ignore", and the missing score is the absolute value of the missing bin's output. The feature order is determined by the observed score. This setting allows the user to separate the importance of a feature into missing and observed components, similar to computing feature importances for a model after using the missing indicator method [112].

Below we demonstrate the non-default options for our trained dnamite model.

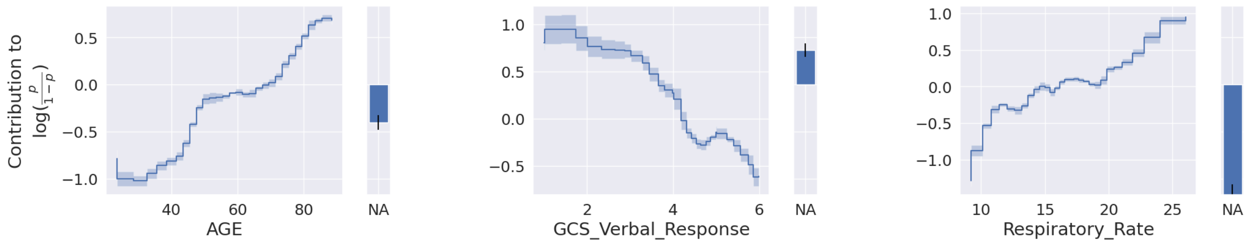
```
model.plot_feature_importances(missing_bin="ignore")
model.plot_feature_importances(missing_bin="stratify")
```



Respiratory rate is the most important feature using the default combined score, but is only the fourth most important feature by observed score. The stratified importance plot sheds light on this difference: respiratory rate has a very large missing score, thereby increasing its overall importance score. This example illustrates how the stratified importance plot can help users better understand the role of each feature. Importantly, such stratification is possible due to DNAMite's embedding module, and cannot be achieved with other NAM architectures.

Next, we use the `plot_shape_function` function to plot estimated shape functions for selected features.

```
model.plot_shape_function([
    "AGE",
    "GCS_Verbal_Response",
    "Respiratory_Rate",
], plot_missing_bin=True)
```



Similar to the feature importance plot, the error region in these shape functions represents pointwise confidence intervals. The function's first parameter accepts either a single string or a list of strings representing the name(s) of the feature(s) to plot. A second optional parameter `plot_missing_bin` defaults to `False` but can be set to `True` to visualize the missing bin score.

Age and respiratory rate, both critical indicators of a patient's health, show positive correlations with in-hospital mortality. Conversely, GCS verbal response exhibits a negative correlation with in-hospital mortality, indicating that better verbal responses are associated with lower mortality rates. Additionally, patients with a missing respiratory rate measurement are at much lower risk of in-hospital mortality. This missing bin score exemplifies informative missingness, as patients who are critically ill are more likely to have their respiratory rate monitored.

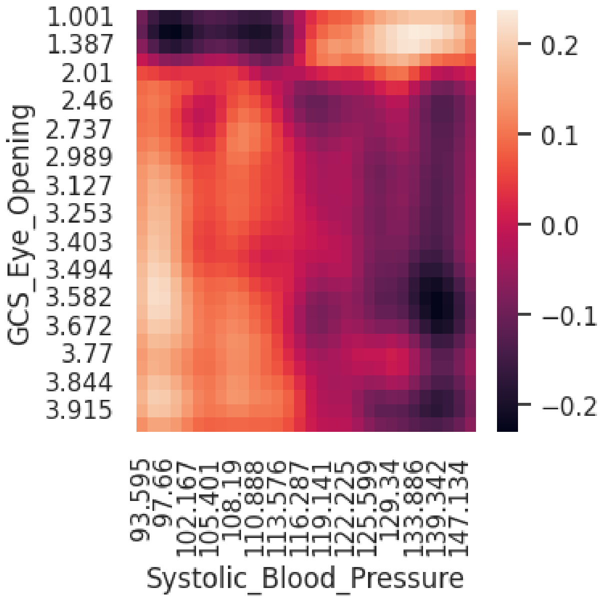
We can also visualize a model's feature interactions. We first list the interactions used by the model, which are stored in `model.selected_pairs_`.

```
model.selected_pairs_
```

```
[['GCS_Eye_Opening', 'Systolic_Blood_Pressure'],
 ['Oxygen_Saturation', 'pH'],
 ['Systolic_Blood_Pressure', 'pH'],
 ['AGE', 'Temperature'],
 ['Heart_Rate', 'pH']]
```

We can plot one of these interactions using the `plot_pair_shape_function` function.

```
model.plot_pair_shape_function(
    "GCS_Eye_Opening",
    "Systolic_Blood_Pressure"
)
```



This interaction plot between eye opening and systolic blood pressure (SBP) shows two regions with increased risk: very low eye opening with mid-to-high SBP as well as low SBP and mid-to-high eye opening. However, this interaction has relatively low importance: it was not a top 10 feature in any of our feature importance plots. Interestingly, we have found that dnamite models without any interactions often still have excellent predictive performance.

**Comparison to Existing Packages**

Figure 3.2 compares the shape function for age among dnamite and three other packages from Table 3.1: interpretml, PyGAM, and mgcv. Both dnamite and interpretml produce similar shape functions with comparable test AUC values. mgcv's shape function is similar in overall shape but is slightly smoother and has a slightly

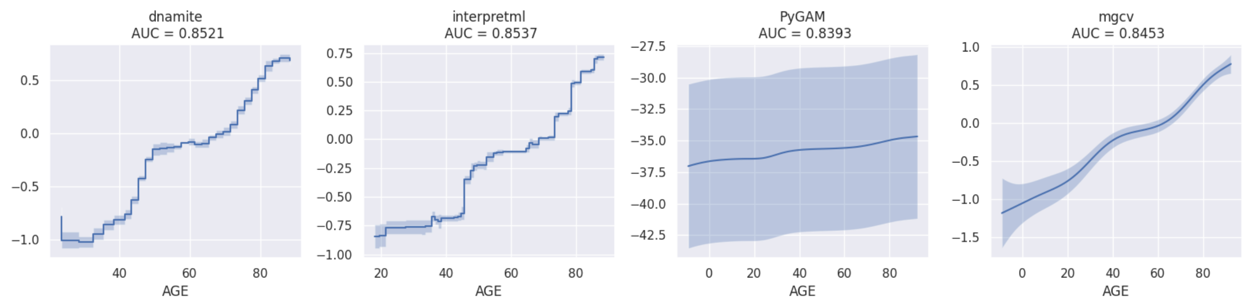


Figure 3.2: Comparison of dnamite to other additive model packages. For each package, the shape function for age is given as well as the test AUC.

worse test AUC. Overall, these three packages provide similar value for standard regression/classification modeling. In contrast, PyGAM generates a shape function that is completely uninterpretable despite having a test AUC that is only marginally worse than the others.

### 3.3.3 Feature Selection

When feature selection is desired, dnamite allows users to select a subset of features before training a model. dnamite performs feature selection as a separate initial step, ensuring consistent feature use across all single split models.

We demonstrate dnamite's ability to do feature selection using the complete cohort dataset.

```
cohort_data = cohort_data.drop([
   "SUBJECT_ID",
   "DOB",
   "DOD",
   "ADMITTIME",
   "DEATH_TIME",
   "CENSOR_TIME",
   "LAST_DISCHTIME",
   "index_time"
], axis=1)
X_cls = cohort_data.drop(["HOSPITAL_EXPIRE_FLAG", "event", "time"], axis=1)
y_cls = cohort_data["HOSPITAL_EXPIRE_FLAG"]
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(
   X_cls, y_cls, test_size=0.2, random_state=seed
)
```

Since this dataset contains 757 features, feature selection is required to obtain a low-dimensional additive model. After initializing a `DNAMiteBinaryClassifier` model as before, we use the `select_features` function to select features and interactions.

```python
from dnamite.models import DNAMiteBinaryClassifier
model = DNAMiteBinaryClassifier(
   random_state=seed,
   device=device,
)
model.select_features(
   X_train_cls,
   y_train_cls,
   reg_param=0.2,
   select_pairs=True,
   pair_reg_param=0.0015,
)
```

Similar to the `fit` function, the `select_features` function accepts training features and labels as its first two arguments. A third required parameter `reg_param` controls the strength of the feature sparsity regularizer, i.e. $\lambda$ in Equation 2.15. Larger values of $\lambda$ will result in fewer selected features. Additionally, `select_features` accepts the following optional parameters that are used to further control the feature selection process.

- `select_pairs`: whether or not to select a list of interactions along with the selected features. Defaults to `False`.
- `pair_reg_param`: similar to `reg_param` but controlling the strength of pair selection.
- `gamma`: controls the steepness of the smooth-step function as described in Equation 2.13. Smaller values reduce the number of iterations required before feature pruning starts. The default value, which we have found works well in many settings, is

$$\texttt{gamma} = \min\left(\frac{N}{B} \cdot \frac{1}{250} \cdot \frac{16}{d}, 1\right), \tag{3.1}$$

  where $N$ is the number of training samples, $B$ is the batch size, and $d$ is the hidden dimension. The intuition is that as $N/B$ (number of iterations per epoch) increases, the model will usually run more iterations before convergence. Hence `gamma` should also be increased so that features are not pruned too early.
- `pair_gamma`: similar to `gamma` but for pairs selection. Defaults to `gamma / 4`.

After feature selection, the selected features and pairs are stored in `model.selected_feats_` and `model.selected_pairs_` respectively.

```python
print("Number of selected features:", len(model.selected_feats_))
print(model.selected_feats_)
```

```
Number of selected features: 12
['AGE',
 'lab_Lactate_mmol/L',
 'lab_Anion_Gap_mEq/L',
 'lab_Bicarbonate_mEq/L',
 'lab_Sodium_mEq/L',
 'lab_RDW_%',
 'chart_Level_of_Conscious',
 'chart_Activity',
 'chart_Code_Status',
 'chart_Gag_Reflex',
 'chart_Orientation',
 'chart_Spontaneous_Movement']
```

```python
print("Number of selected pairs:", len(model.selected_pairs_))
print(model.selected_pairs_)
```

```
Number of selected pairs: 1
[['lab_Lactate_mmol/L', 'lab_Bicarbonate_mEq/L']]
```

Now, we call the `fit` function as before, and the model will automatically use the selected features and interactions.

```python
model.fit(X_train_cls, y_train_cls)
```

We obtain model predictions similarly, with the model again automatically using the selected features and interactions.

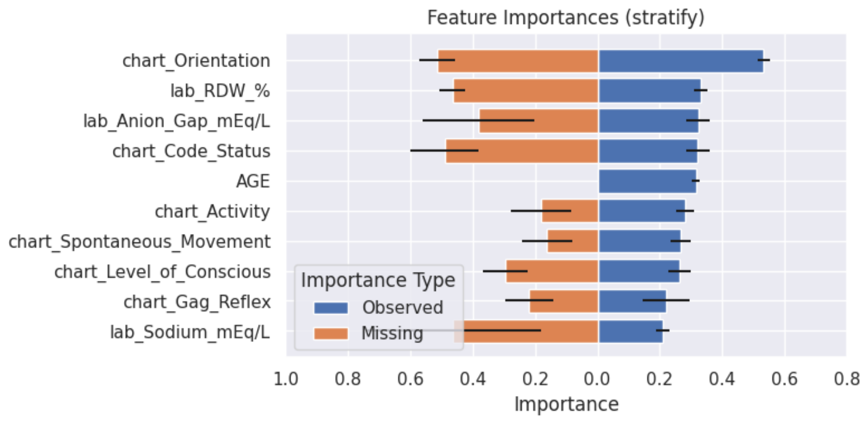
```python
from sklearn.metrics import roc_auc_score
preds = model.predict_proba(X_test_cls)
selection_auc = np.round(roc_auc_score(y_test_cls, preds), 4)
print(f"AUC with automated feature selection: {selection_auc}")
print(f"AUC with manual feature selection: {dnamite_auc}")
```

```
AUC with automated feature selection: 0.8792
AUC with manual feature selection: 0.8521
```

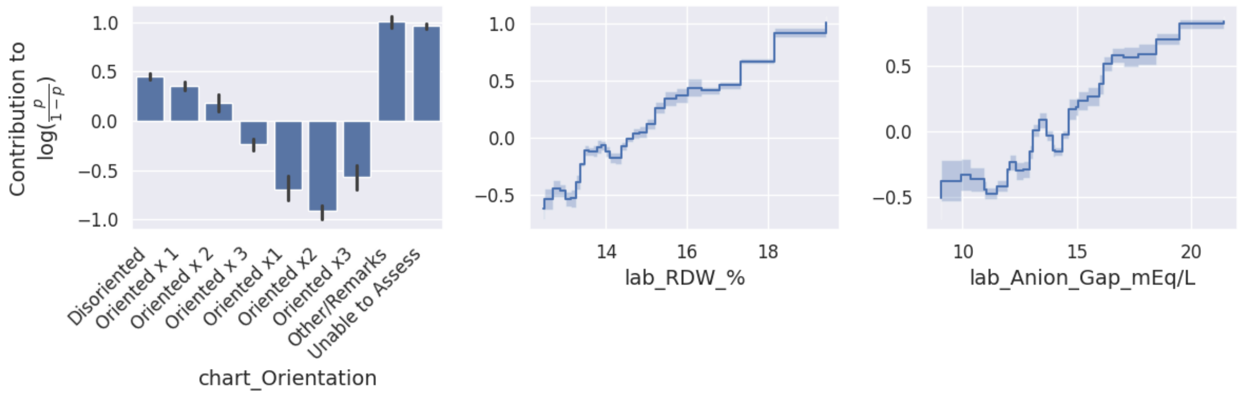
Compared to the features selected in the benchmark, the AUC of this model is approximately 0.025 higher, demonstrating the potential lift in predictive performance due to automated feature selection. We see the benefits of combining feature selection and additive modeling within the same package and model structure, a combination that distinguishes dnamite from previous packages for additive modeling or feature selection.

As before, we can visualize feature importances and shape functions after training.

```python
model.plot_feature_importances(missing_bin="stratify")
```



```
model.plot_shape_function([
    "chart_Level_of_Conscious",
    "chart_Orientation",
    "AGE",
])
```



Interestingly, all of the top features except age are different in this model compared to the benchmark model. Many of these new features represent direct or indirect measures of overall health and/or seriousness of care. For example, the most important feature is orientation, which assesses a patient's cognitive awareness. Patients who are more disoriented, need special remarks, or cannot have their orientation assessed are (unsurprisingly) at higher mortality risk. Without algorithmic feature selection, MIMIC users might never realize that any feature has such strong correlation with in-hospital mortality. Moreover, if a user wishes to remove a feature like orientation, believing that it may leak information about the label, they can easily fit a new dnamite model that excludes that feature. Users can also choose to combine manual and algorithmic feature selection to refine their model.

**Comparison to Existing Packages**

As illustrated in Table 3.1, the only other additive model package that supports feature selection is PiML. However, PiML's feature selection capabilities are notably inferior to those of dnamite. Firstly, GAMI-Net

[123] is the only model in PiML that implements feature selection. GAMI-Net initially fits all features and subsequently selects the top $k$ features based on shape function variation. This strategy is inefficient because the model must fully estimate shape functions for all features before performing any selection. In contrast, a dnamite model can prune a feature as soon as the feature's gate is reduced to zero. Additionally, GAMI-Net's approach to feature selection is more heuristic compared to dnamite's, as dnamite integrates feature selection directly into the optimization process. Furthermore, dnamite's feature selection has been demonstrated to outperform GAMI-Net [44]. PiML also lacks comprehensive user guides or examples to illustrate how feature selection can be executed using GAMI-Net; the PiML API includes a function, `fine_tune_selected`, for pruning features but does not provide examples of its application.

## 3.3.4 Survival Analysis

Training and analyzing a dnamite model for survival analysis is similar to the workflow for regression or classification. Following scikit-survival [80], dnamite expects labels prepared as a structured NumPy array with dtype `[('event', 'bool'), ('time', 'float')]`. The `event` field should store `True` when the event of interest is observed and `False` when the sample is censored. The `time` field should store the time-to-event for observed samples and time-to-censoring for censored samples. For the MIMIC mortality data, patients are censored with their last discharge time if their death time is not recorded in the dataset.

```
X_surv = cohort_data.drop(
   ["HOSPITAL_EXPIRE_FLAG", "event", "time"],
   axis=1
)
y_surv = np.empty(
   dtype=[('event', 'bool'), ('time', 'float64')],
   shape=(cohort_data.shape[0],)
)
y_surv["event"] = cohort_data["event"].values
y_surv["time"] = np.clip(cohort_data["time"].values, 1e-5, np.inf)
X_train_surv, X_test_surv, y_train_surv, y_test_surv = train_test_split(
   X_surv, y_surv, test_size=0.2, random_state=seed
)
```

dnamite uses the class `DNAMiteSurvival` for survival analysis models. We can fit a feature-sparse dnamite survival model using the same functions as before.

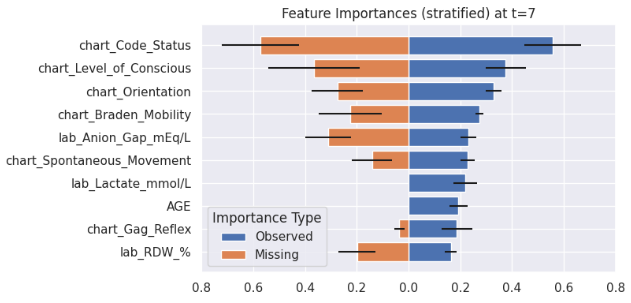
```
from dnamite.models import DNAMiteSurvival
model = DNAMiteSurvival(
    random_state=seed,
    device=device,
)
model.select_features(
    X_train_surv,
    y_train_surv,
    select_pairs=True,
    reg_param=0.75,
    pair_reg_param=0.005,
)
print("Number of selected features:", len(model.selected_feats_))
print("Number of selected pairs:", len(model.selected_pairs_))
model.fit(X_train_surv, y_train_surv)
```

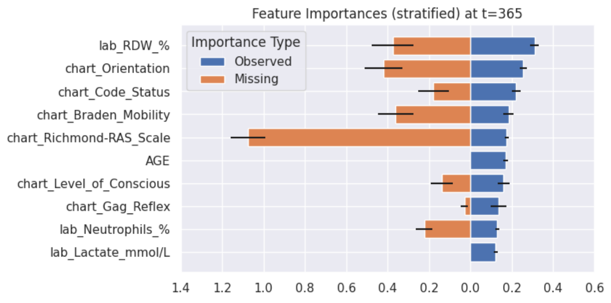
```
Number of selected features: 17
Number of selected pairs: 4
```

dnamite survival models optimize the IPCW loss given in Equation 2.18. By default, the censoring distribution is estimated using a Kaplan-Meier model. To use a Cox model instead, the optional parameter `censor_estimator` should be set to `"cox"` at initialization.

Unlike previous classification models, a dnamite survival model yields interpretations for specific evaluation times. In the `plot_feature_importances` function, we add a parameter `eval_times` to specify either a single evaluation time or a list of evaluation times for which feature importance scores should be plotted.

```
model.plot_feature_importances(eval_times=[7, 365], missing_bin="stratify")
```

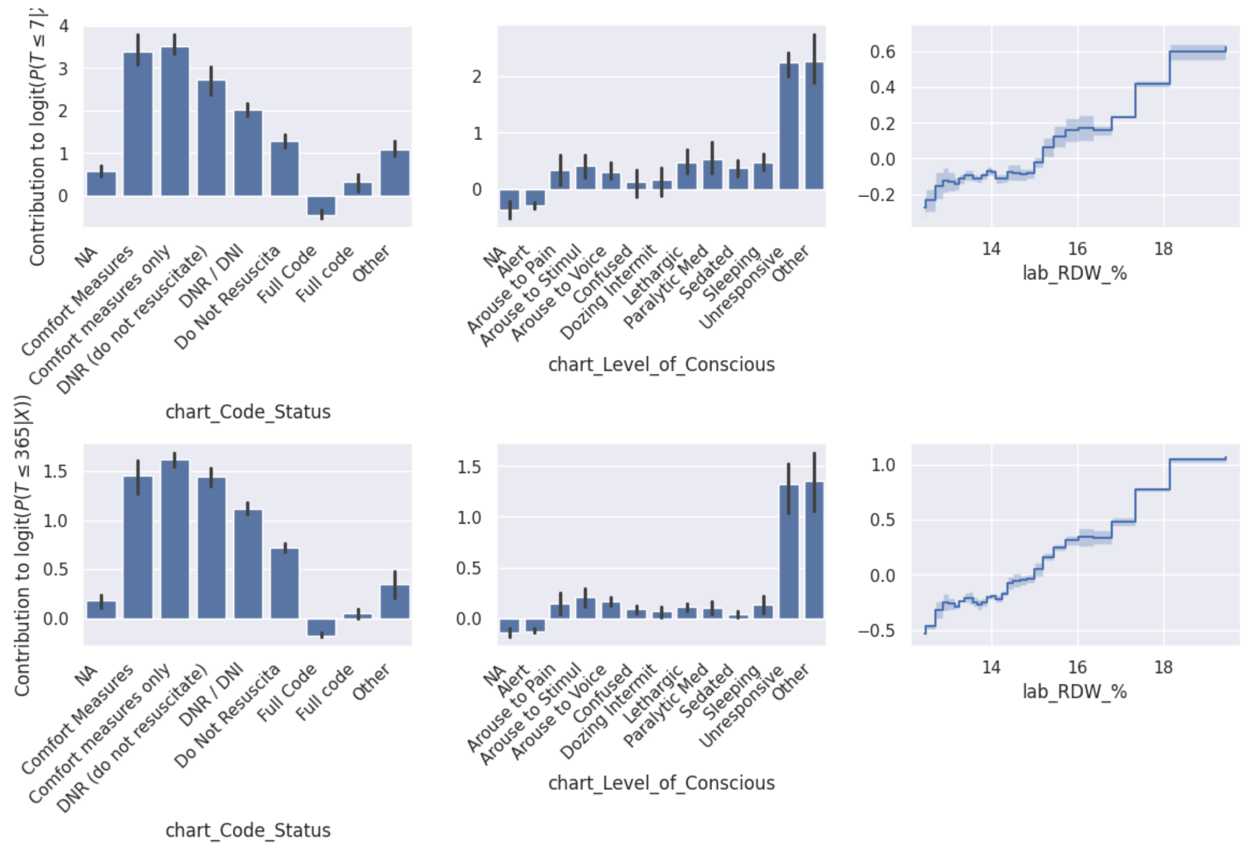




These importances illustrate interesting differences in the risk factors for short-term (next 7 days) and long-term (next year) mortality risk. For short-term mortality, the most important feature is code status, which describes how much medical intervention should be taken in the event of a medical emergency. Another important feature relative is Anion Gap, with elevated levels being a well-documented predictor of imminent mortality risk [14]. Conversely, Red Cell Distribution Width (RDW) is relatively more significant for predicting long-term mortality risk [116].

Similarly, we call the `plot_shape_function` function with parameter `eval_times` to plot shape functions for features at specific evaluation times.

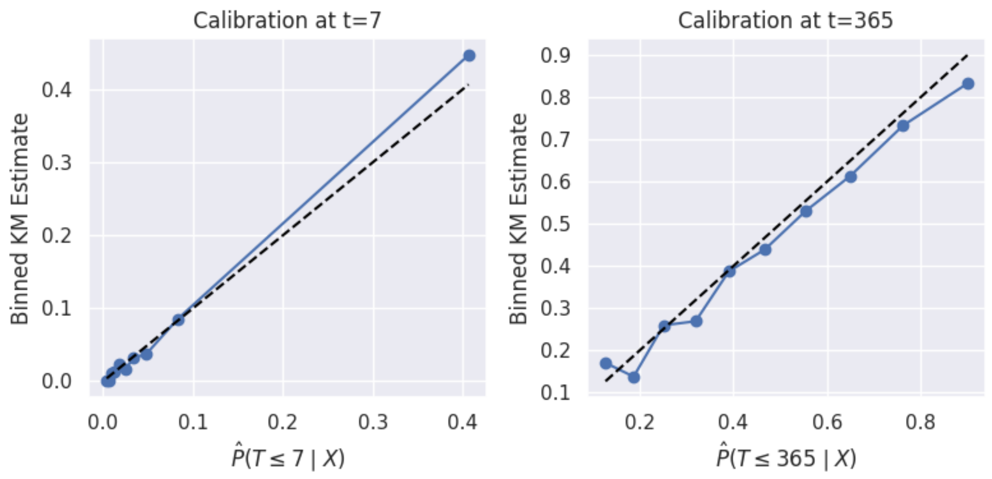
```
model.plot_shape_function(
    ["chart_Code_Status", "chart_Level_of_Conscious", "lab_RDW_%"],
    eval_times=[7, 365]
)
```



The patterns of these shape functions remain consistent at different evaluation times, although their magnitudes change to reflect varying levels of importance across time.

Assessing the calibration of survival models is important as calibration ensures that survival predictions have valid probabilistic interpretations [5]. dnamite supports calibration assessment via calibration plots. To make these calibration plots, dnamite first partitions samples into bins using quantiles of the predicted CDF estimates at a given evaluation time. Within each bin, a Kaplan-Meier estimator is used to estimate the true CDF score, which is plotted against the bin's average predicted CDF from the model. A trained dnamite survival model can call the function `make_calibration_plot` to assess calibration. The function accepts features, labels, and a list of evaluation times.

```
model.make_calibration_plot(X_test_surv, y_test_surv, eval_times=[7, 365])
```



These plots illustrate that the trained model is reasonably calibrated since the points are all close to the $y = x$ line. If the model were to appear uncalibrated, a possible cause is that $C \not\perp\!\!\!\perp X$, so that the IPCW loss in Equation 2.18 is improper. In that case, we recommend retraining the model setting `censor_estimator` to `"cox"` to see if relaxing the assumption to $C \perp\!\!\!\perp T \mid X$ improves calibration.

**Comparison to Existing Packages**

As shown in Table 3.1, there are no existing Python packages that support training additive models for survival analysis. The R packages gam and mgcv support survival analysis, but with several limitations compared to dnamite. First, both packages only support training with the Cox proportional hazards loss, constraining these models to the proportional hazards assumption that dnamite avoids. Second, neither R package supports feature selection, which is a core component of dnamite. Third, neither package can handle missing values natively. Instead, they would require missing value imputation, and so they cannot identify the impact of missing values in feature importances and shape functions.

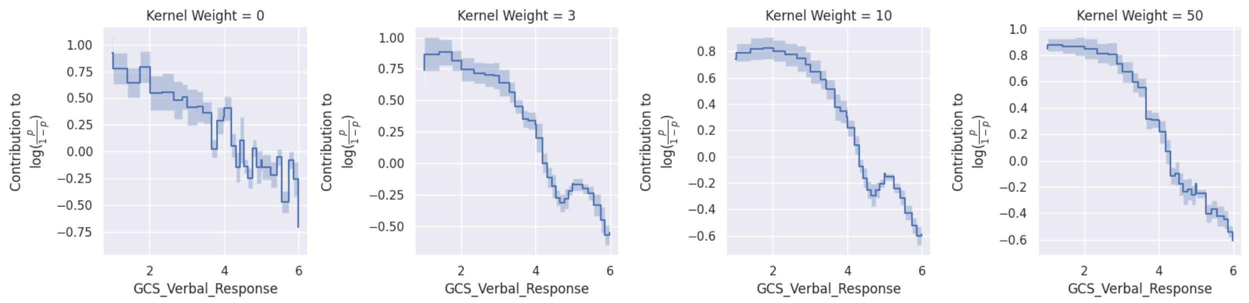
## 3.3.5 Advanced Usage

**Controlling Smoothness**

The parameter $\gamma$ in Algorithm 2.5 controls the smoothness of dnamite's shape functions. Exposing $\gamma$ as an optional parameter gives users more control over shape function smoothness. A user who wants simple shape functions should set $\gamma$ to a large value. Conversely, a user who wants more complicated shape functions at the risk of overfitting should set $\gamma$ to a small value.

The optional parameter `kernel_weight` sets $\gamma$ in dnamite models at initialization. Below we demonstrate the impact of changing `kernel_weight` on the `GCS_Verbal_Response` feature in the benchmark binary classification dataset.

```
fig, axes = plt.subplots(1, 5, figsize=(20, 4))
kernel_weights = [0, 1, 5, 10, 100]
for kw, ax in zip(kernel_weights, axes):
    model = DNAMiteBinaryClassifier(
        random_state=seed,
        device=device,
        kernel_weight=kw,
    )
    model.fit(X_train_benchmark, y_train_benchmark)
    model.plot_shape_function("GCS_Verbal_Response", axes=[ax])
    ax.set_title(f"Kernel Weight = {kw}")
plt.tight_layout()
```



When $\phi = 0$, kernel smoothing is completely removed, and the shape function is (as expected) very noisy. With $\phi = 3$ and $\phi = 10$, a smoother shape emerges but still with two inflection points. Finally, when $\phi = 50$, the shape function becomes nearly monotonic. The default value is 3, and we recommend choosing a value between 1 and 10 for most use cases.

### Monotonic Constraints

Increasing $\phi$ gives dnamite users one way to simplify model shape functions. Alternatively, users may insist that shape functions are fully monotonic due to prior domain knowledge of the problem. In these cases, dnamite allows users to provide monotonic constraints on individual features. dnamite follows the interpretml API for user-supplied monotonic constraints: a single list of length `n_features` is required, where 0 indicates no constraints, 1 indicates a monotone increasing constraint, and -1 indicates a monotone decreasing constraint. Below we train a dnamite model with selected monotone features by passing the above list to the `monotone_constraints` optional parameter.

```
monotone_constraints = [
    1 if c == "AGE" else
    -1 if c in ["GCS_Eye_Opening", "GCS_Verbal_Response"] else
    0
    for c in X_train_benchmark.columns
]
model = DNAMiteBinaryClassifier(
    random_state=seed,
    device=device,
    monotone_constraints=monotone_constraints,
)
model.fit(X_train_benchmark, y_train_benchmark)
```

dnamite implements monotonic constraints as follows. Suppose the input for feature $j$ falls in bin index $i$. Without monotonic constraints, the output for feature $j$ would be $\hat{f}_j(i)$. For a monotone increasing feature, this output is replaced with $o_j + \sum_{k \le i} \hat{f}_j(i-k)^2$, where $o_j$ is a learnable offset parameter. Similarly, for a monotone decreasing feature, the output is replaced with $o_j - \sum_{k \le i} \hat{f}_j(i-k)^2$.

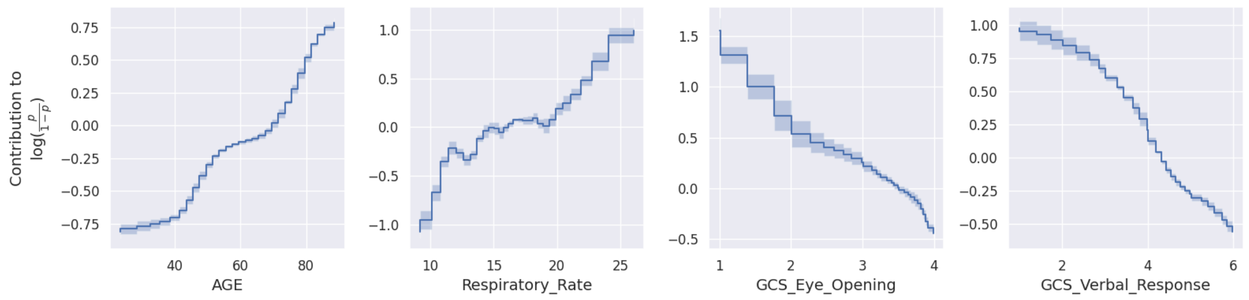
Below we compare the AUC of this model to the model trained in 3.3.2, which has no monotonicity constraints.

```
preds = model.predict_proba(X_test_benchmark)
monotone_constraints_auc = roc_auc_score(y_test_benchmark, preds)
print(f"Monotone constraints AUC: {round(monotone_constraints_auc, 4)}")
print(f"Baseline AUC: {dnamite_auc}")
```

```
Monotone constraints AUC: 0.8484
Baseline AUC: 0.8521
```

The monotonicity constraints have little effect on the predictive performance. Further, we can plot shape functions to verify that the monotonic constraints are working as intended.

```
model.plot_shape_function([
    "AGE",
    "Respiratory_Rate",
    "GCS_Eye_Opening",
    "GCS_Verbal_Response",
])
```



The shape functions for age, GCS eye opening, and GCS verbal response follow their intended monotonicity constraints, while the respiratory rate shape function is not monotonic.

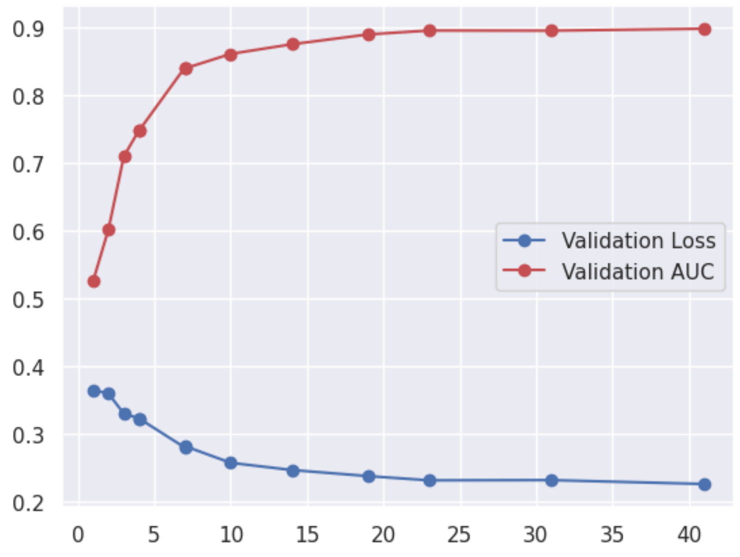
### Feature Selection Paths

Determining the right number of features to choose can be challenging in feature selection models. dnamite controls the number of features chosen through `reg_param` ($\lambda$ in Equation 2.15). One strategy sets $\lambda$ via hyperparameter optimization using validation predictive performance. However, this strategy will prioritize accuracy over interpretability, producing models with (possibly too) many features.

An alternative approach is to test multiple $\lambda$ values and select the smallest feature set that maintains competitive predictive performance. To facilitate this approach, dnamite implements a function `get_regularization_path` that helps users select the right number of features for their dataset. The function tries several $\lambda$ values and reports the validation loss and score associated with each value.

```
model = DNAMiteBinaryClassifier(
    random_state=seed,
    device=device,
)
feats, path_data = model.get_regularization_path(
    X_train_cls, y_train_cls, init_reg_param=0.01
)
plt.plot(
    path_data["num_feats"], path_data["val_loss"],
    marker='o', color='b'
)
plt.plot(
    path_data["num_feats"], path_data["val_AUC"],
    marker='o', color='r'
)
plt.legend(["Validation Loss", "Validation AUC"])
```

```
<matplotlib.legend.Legend at 0x7f80dbec7e50>
```



Selecting less than 10 features yields significantly worse predictive performance, while the performance improvements after selecting about 15 features are marginal. To see the features that are selected at each point in this graph, the returned dictionary `feats` can be queried using the number of selected features as the key.

```
feats[10]
```

```
['AGE',
 'lab_Lactate_mmol/L',
 'lab_Anion_Gap_mEq/L',
 'lab_Bicarbonate_mEq/L',
 'lab_Sodium_mEq/L',
 'lab_RDW_%',
 'chart_Level_of_Conscious',
 'chart_Activity',
 'chart_Code_Status',
 'chart_Orientation']
```

## 3.4 Conclusion

This chapter introduces dnamite [114], a Python package that implements a suite of Neural Additive Models (NAMs). dnamite's models can be used for regression, classification, and survival analysis, offering improved accuracy compared to linear models and improved interpretability compared to black-box models. Further, dnamite supports automated feature selection before training, allowing users to fit low-dimensional additive models on high-dimensional datasets. By using the DNAMite architecture, dnamite's models can flexibly adapt to different desired smoothness levels in shape functions, as well as elegantly handle categorical features and missing values. Our usage demonstration illustrates the power of dnamite as a package for interpretable machine learning.

# Concluding Remarks

Throughout this dissertation, we have presented novel methodologies for interpretable machine learning (ML) with applications in healthcare, focusing particularly on handling missing values and developing explainable models for survival analysis. As we look to the future of ML in healthcare, the approaches developed here offer several promising directions for continued research and practical implementation.

The investigation of missing value handling in Chapter 1 demonstrates that the Missing Indicator Method (MIM) and Selective MIM (SMIM) provide effective strategies for leveraging the signal in missing patterns. This work challenges the conventional wisdom that missing values are merely obstacles to overcome and instead reveals them as potential sources of predictive information. Future research could extend these approaches to time-series data in healthcare, where patterns of missingness across time may contain even richer signals about patient trajectories and outcomes.

The interpretable survival analysis models presented in Chapter 2, including survival stacking, DyS, and DNAMite, represent advances in our ability to predict time-to-event outcomes while maintaining full model transparency. As healthcare increasingly focuses on personalized medicine and risk stratification, these methods offer clinicians interpretable predictions that can directly inform treatment decisions and resource allocation. Looking forward, these approaches could be extended to incorporate multimodal data including genomics, imaging, and unstructured text, while preserving their essential interpretability.

The dnamite Python package described in Chapter 3 bridges the gap between methodological innovation and practical implementation, making neural additive models accessible for researchers and practitioners. This tool represents an important step toward increasing the accessibility of interpretable ML approaches. We hope to continuously improve the package as more exciting clinical applications arise.

As ML continues to evolve, the tension between model complexity and interpretability remains a central challenge, particularly in healthcare. The methodologies presented in this dissertation demonstrate that this tension can be productively resolved, as powerful predictive models need not sacrifice transparency. Moving forward, interpretable ML approaches will likely play an increasingly important role in healthcare as regulatory frameworks around AI continue to emphasize explainability and as clinicians demand models whose predictions they can trust and understand.

The future of interpretable machine learning in healthcare will likely involve closer collaboration between ML researchers, healthcare providers, and patients themselves. Models that cannot only make accurate predictions but also communicate their reasoning effectively will be essential for building trust and driving adoption. By continuing to advance methods that balance predictive power with interpretability, we can develop machine learning systems that serve as valuable partners in healthcare decision-making rather than inscrutable black boxes.

In conclusion, this dissertation has contributed novel methodologies for handling missing data and building interpretable survival models, with a focus on healthcare applications. These contributions lay the groundwork for future innovations in interpretable machine learning that can meaningfully improve healthcare delivery and patient outcomes while maintaining the transparency necessary for responsible implementation in clinical settings.

# Bibliography

[1] Talal AA Abdullah, Mohd Soperi Mohd Zahid, and Waleed Ali. A review of interpretable ml in healthcare: taxonomy, applications, challenges, and future directions. *Symmetry*, 13(12):2439, 2021.

[2] Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E Hinton. Neural additive models: Interpretable machine learning with neural nets. *Advances in neural information processing systems*, 34:4699–4711, 2021.

[3] Abdallah Alabdallah, Sepideh Pashami, Thorsteinn Rögnvaldsson, and Mattias Ohlsson. Survshap: a proxy-based algorithm for explaining survival models with shap. In *2022 IEEE 9th international conference on data science and advanced analytics (DSAA)*, pages 1–10. IEEE, 2022.

[4] Abdul Fatir Ansari, Lorenzo Stella, Caner Turkmen, Xiyuan Zhang, Pedro Mercado, Huibin Shen, Oleksandr Shchur, Syama Sundar Rangapuram, Sebastian Pineda Arango, Shubham Kapoor, et al. Chronos: Learning the language of time series. *arXiv preprint arXiv:2403.07815*, 2024.

[5] Peter C Austin, Frank E Harrell Jr, and David van Klaveren. Graphical calibration curves and the integrated calibration index (ici) for survival models. *Statistics in Medicine*, 39(21):2714–2742, 2020.

[6] Anand Avati, Tony Duan, Sharon Zhou, Kenneth Jung, Nigam H Shah, and Andrew Y Ng. Countdown regression: sharp and calibrated survival predictions. In *Uncertainty in Artificial Intelligence*, pages 145–155. PMLR, 2020.

[7] Amanda N Baraldi and Craig K Enders. An introduction to modern missing data analyses. *Journal of school psychology*, 48(1):5–37, 2010.

[8] Brett K Beaulieu-Jones, Patryk Orzechowski, and Jason H Moore. Mapping patient trajectories using longitudinal extraction and deep learning in the mimic-iii critical care database. In *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2018: Proceedings of the Pacific Symposium*, pages 123–132. World Scientific, 2018.

[9] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological)*, 57(1):289–300, 1995.

[10] Dimitris Bertsimas, Arthur Delarue, and Jean Pauphilet. Prediction with missing data. *arXiv preprint arXiv:2104.03158*, 2021.

[11] B Bozkurt, T Ahmad, K Alexander, W L Baker, K Bosak, K Breathett, S Carter, M H Drazner, S M Dunlay, G C Fonarow, S J Greene, P Heidenreich, J E Ho, E Hsich, N E Ibrahim, L M Jones, S S Khan, P Khazanie, T Koelling, C S Lee, A A Morris, 2nd Page, R L, A Pandey, M R Piano, A T Sandhu, J Stehlik, L W Stevenson, J Teerlink, A R Vest, C Yancy, and B; WRITING COMMITTEE MEMBERS Ziaeian. Hf stats 2024: Heart failure epidemiology and outcomes statistics an updated 2024 report from the heart failure society of america. *J Card Fail*, 31(1):66–116, Jan 2025.

[12] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[13] Leo Breiman. *Classification and regression trees.* Routledge, 2017.

[14] Barry E Brenner. Clinical significance of the elevated anion gap. *The American journal of medicine*, 79(3):289–296, 1985.

[15] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1721–1730, 2015.

[16] Alex J Chang, Yilin Liang, Michael P Girouard, Ankeet S Bhatt, Alexander T Sandhu, Andrew J Sauer, Stephen J Greene, Josephine Harrington, Alan S Go, and Andrew P Ambrosy. Changing the paradigm in heart failure: shifting from treatment to prevention. *Heart Failure Reviews*, 30(1):177–189, 2025.

[17] Chun-Hao Chang, Rich Caruana, and Anna Goldenberg. Node-gam: Neural generalized additive model for interpretable deep learning. *arXiv preprint arXiv:2106.01613*, 2021.

[18] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[19] Taane G Clark, Michael J Bradburn, Sharon B Love, and Douglas G Altman. Survival analysis part i: basic concepts and first analyses. *British journal of cancer*, 89(2):232–238, 2003.

[20] David R Cox. Regression models and life-tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 34(2):187–202, 1972.

[21] Erin Craig, Chenyang Zhong, and Robert Tibshirani. Survival stacking: casting survival analysis as a classification problem. *arXiv preprint arXiv:2107.13480*, 2021.

[22] Jacob Deasy, Pietro Liò, and Ari Ercole. Dynamic survival prediction in intensive care units from heterogeneous time series without the need for variable selection or curation. *Scientific Reports*, 10(1):22129, 2020.

[23] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

[24] Ning Ding, Cuirong Guo, Changluo Li, Yang Zhou, and Xiangping Chai. An artificial neural networks model for early predicting in-hospital mortality in acute pancreatitis in mimic-iii. *BioMed research international*, 2021, 2021.

[25] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-Sklearn 2.0: Hands-free automl via meta-learning. *arXiv preprint arXiv:2007.04074*, 2020.

[26] Unai Garciarena and Roberto Santana. An extensive analysis of the interaction between missing data types, imputation methods, and supervised classifiers. *Expert Systems with Applications*, 89:52–65, 2017.

[27] Unai Garciarena, Roberto Santana, and Alexander Mendiburu. Evolving imputation strategies for missing data in classification problems with TPOT. *arXiv preprint arXiv:1706.01120*, 2017.

[28] Thanos Gentimis, Alnaser Ala'J, Alex Durante, Kyle Cook, and Robert Steele. Predicting hospital length of stay using neural networks on mimic iii data. In *2017 IEEE 15th intl conf on dependable, autonomic and secure computing, 15th intl conf on pervasive intelligence and computing, 3rd intl conf on big data intelligence and computing and cyber science and technology congress (DASC/PiCom/DataCom/CyberSciTech)*, pages 1194–1201. IEEE, 2017.

[29] Tilmann Gneiting, Fadoua Balabdaoui, and Adrian E Raftery. Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 69(2):243–268, 2007.

[30] Lovedeep Gondara and Ke Wang. Mida: Multiple imputation using denoising autoencoders. In *Advances in Knowledge Discovery and Data Mining: 22nd Pacific-Asia Conference, PAKDD 2018, Melbourne, VIC, Australia, June 3-6, 2018, Proceedings, Part III 22*, pages 260–272. Springer, 2018.

[31] Yura Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. *arXiv preprint arXiv:2203.05556*, 2022.

[32] Brandon M Greenwell et al. pdp: An r package for constructing partial dependence plots. *R J.*, 9(1):421, 2017.

[33] Rolf HH Groenwold, Ian R White, A Rogier T Donders, James R Carpenter, Douglas G Altman, and Karel GM Moons. Missing covariate data in clinical research: when and when not to use the missing-indicator method for analysis. *Cmaj*, 184(11):1265–1269, 2012.

[34] Bruce Hansen. *Econometrics*. Princeton University Press, 2022.

[35] Hrayr Harutyunyan, Hrant Khachatrian, David C. Kale, Greg Ver Steeg, and Aram Galstyan. Multitask learning and benchmarking with clinical time series data. *Scientific Data*, 6(1):96, 2019.

[36] Trevor Hastie. *gam: Generalized Additive Models*, 2024. R package version 1.22-5.

[37] Trevor Hastie and Robert Tibshirani. Generalized additive models. *Statistical Science*, pages 297–310, 1986.

[38] Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In *International Conference on Machine Learning*, pages 4138–4148. PMLR, 2020.

[39] Benjamin Hofner, Andreas Mayr, Nikolay Robinzonov, and Matthias Schmid. Model-based boosting in r: a hands-on tutorial using the r package mboost. *Computational statistics*, 29:3–35, 2014.

[40] Benjamin Hofner, Andreas Mayr, and Matthias Schmid. gamboostlss: An r package for model building and variable selection in the gamlss framework. *Journal of Statistical Software*, 74:1–31, 2016.

[41] James Honaker, Gary King, and Matthew Blackwell. Amelia II: A program for missing data. *Journal of statistical software*, 45:1–47, 2011.

[42] Shi Hu, Egill Fridgeirsson, Guido van Wingen, and Max Welling. Transformer-based deep survival analysis. In *Survival Prediction-Algorithms, Challenges and Applications*, pages 132–148. PMLR, 2021.

[43] Xinyu Hu, Tanmay Binaykiya, Eric Frank, and Olcay Cirit. Deepreta: an eta post-processing system at scale. *arXiv preprint arXiv:2206.02127*, 2022.

[44] Shibal Ibrahim, Gabriel Afriat, Kayhan Behdin, and Rahul Mazumder. Grand-slamin'interpretable additive modeling with structural constraints. *Advances in Neural Information Processing Systems*, 36, 2024.

[45] Shibal Ibrahim, Gabriel Isaac Afriat, Kayhan Behdin, and Rahul Mazumder. Grand-slamin'interpretable additive modeling with structural constraints. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[46] Niels Bruun Ipsen, Pierre-Alexandre Mattei, and Jes Frellsen. How to deal with missing data in supervised deep learning? In *International Conference on Learning Representations*, 2021.

[47] Hemant Ishwaran, Udaya B Kogalur, Eugene H Blackstone, and Michael S Lauer. Random survival forests. 2008.

[48] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[49] Julie Josse, Nicolas Prost, Erwan Scornet, and Gaël Varoquaux. On the consistency of supervised learning with missing values. *arXiv preprint arXiv:1902.06931*, 2019.

[50] Fahad Kamran and Jenna Wiens. Estimating calibrated individualized survival curves with deep learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 240–248, 2021.

[51] Sadiya S Khan, Josef Coresh, Michael J Pencina, Chiadi E Ndumele, Janani Rangaswami, Sheryl L Chow, Latha P Palaniappan, Laurence S Sperling, Salim S Virani, Jennifer E Ho, et al. Novel prediction equations for absolute risk assessment of total cardiovascular disease incorporating cardiovascular-kidney-metabolic health: a scientific statement from the american heart association. *Circulation*, 148(24):1982–2004, 2023.

[52] Sadiya S Khan, Hongyan Ning, Sanjiv J Shah, Clyde W Yancy, Mercedes Carnethon, Jarett D Berry, Robert J Mentz, Emily O'Brien, Adolfo Correa, Navin Suthahar, et al. 10-year risk equations for incident

heart failure in the general population. *Journal of the American College of Cardiology*, 73(19):2388–2397, 2019.

[53] Mirjam J Knol, Kristel JM Janssen, A Rogier T Donders, Antoine CG Egberts, E Rob Heerdink, Diederick E Grobbee, Karel GM Moons, and Mirjam I Geerlings. Unpredictable bias when using the missing indicator method or complete case analysis for missing confounder values: an empirical example. *Journal of clinical epidemiology*, 63(7):728–736, 2010.

[54] Maxim S Kovalev, Lev V Utkin, and Ernest M Kasimov. Survlime: A method for explaining machine learning survival models. *Knowledge-Based Systems*, 203:106164, 2020.

[55] Mateusz Krzyziński, Mikołaj Spytek, Hubert Baniecki, and Przemysław Biecek. Survshap (t): time-dependent explanations of machine learning survival models. *Knowledge-Based Systems*, 262:110234, 2023.

[56] I Elizabeth Kumar, Suresh Venkatasubramanian, Carlos Scheidegger, and Sorelle Friedler. Problems with shapley-value-based explanations as feature importance measures. In *International Conference on Machine Learning*, pages 5491–5500. PMLR, 2020.

[57] Trent Kyono, Yao Zhang, Alexis Bellot, and Mihaela van der Schaar. MIRACLE: Causally-aware imputation via learning missing data mechanisms. *Advances in Neural Information Processing Systems*, 34, 2021.

[58] Sophie Hanna Langbein, Mateusz Krzyziński, Mikołaj Spytek, Hubert Baniecki, Przemysław Biecek, and Marvin N Wright. Interpretable machine learning for survival analysis. *arXiv preprint arXiv:2403.10250*, 2024.

[59] Marine Le Morvan, Julie Josse, Thomas Moreau, Erwan Scornet, and Gaël Varoquaux. NeuMiss networks: differentiable programming for supervised learning with missing values. *Advances in Neural Information Processing Systems*, 33:5980–5990, 2020.

[60] Marine Le Morvan, Julie Josse, Erwan Scornet, and Gaël Varoquaux. What's a good imputation to predict with missing values? *Advances in Neural Information Processing Systems*, 34, 2021.

[61] Marine Le Morvan, Nicolas Prost, Julie Josse, Erwan Scornet, and Gaël Varoquaux. Linear predictor on linearly-generated data with missing values: non consistency and solutions. In *International Conference on Artificial Intelligence and Statistics*, pages 3165–3174. PMLR, 2020.

[62] Changhee Lee, William Zame, Jinsung Yoon, and Mihaela Van Der Schaar. Deephit: A deep learning approach to survival analysis with competing risks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[63] Benjamin Lengerich, Sarah Tan, Chun-Hao Chang, Giles Hooker, and Rich Caruana. Purifying interaction effects with the functional anova: An efficient algorithm for recovering identifiable additive models. In *International Conference on Artificial Intelligence and Statistics*, pages 2402–2412. PMLR, 2020.

[64] Jia Li, Mengdie Wang, Michael S Steinbach, Vipin Kumar, and Gyorgy J Simon. Don't do imputation: Dealing with informative missing values in ehr data analysis. In *2018 IEEE International Conference on Big Knowledge (ICBK)*, pages 415–422. IEEE, 2018.

[65] Xiyue Liao and Mary C Meyer. cgam: an r package for the constrained generalized additive model. *Journal of Statistical Software*, 89:1–24, 2019.

[66] Wei-Chao Lin and Chih-Fong Tsai. Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review*, 53(2):1487–1509, 2020.

[67] Roderick JA Little. A test of missing completely at random for multivariate data with missing values. *Journal of the American statistical Association*, 83(404):1198–1202, 1988.

[68] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

[69] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631, 2013.

[70] Mattias Luber, Anton Thielmann, and Benjamin Säfken. Structural neural additive models: Enhanced interpretable machine learning. *arXiv preprint arXiv:2302.09275*, 2023.

[71] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[72] Pierre-Alexandre Mattei and Jes Frellsen. Miwae: Deep generative modelling and imputation of incomplete data sets. In *International conference on machine learning*, pages 4413–4423. PMLR, 2019.

[73] W James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Interpretable machine learning: definitions, methods, and applications. *arXiv preprint arXiv:1901.04592*, 2019.

[74] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632, 2005.

[75] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*, 2019.

[76] Siddhartha Nuthakki, Sunil Neela, Judy W Gichoya, and Saptarshi Purkayastha. Natural language processing of mimic-iii clinical notes for identifying diagnosis and procedures with neural networks. *arXiv preprint arXiv:1912.12397*, 2019.

[77] Ines Ortega-Fernandez, Marta Sestelo, and Nora M Villanueva. Explainable generalized additive neural networks with independent neural network training. *Statistics and Computing*, 34(1):6, 2023.

[78] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[79] Alexandre Perez-Lebel, Gaël Varoquaux, Marine Le Morvan, Julie Josse, and Jean-Baptiste Poline. Benchmarking missing-values approaches for predictive models on health databases. *GigaScience*, 11, 2022.

[80] Sebastian Pölsterl. scikit-survival: A library for time-to-event analysis built on top of scikit-learn. *Journal of Machine Learning Research*, 21(212):1–6, 2020.

[81] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*, 2019.

[82] Forough Poursabzi-Sangdeh, Daniel G Goldstein, Jake M Hofman, Jennifer Wortman Wortman Vaughan, and Hanna Wallach. Manipulating and measuring model interpretability. In *Proceedings of the 2021 CHI conference on human factors in computing systems*, pages 1–52, 2021.

[83] Sanjay Purushotham, Chuizheng Meng, Zhengping Che, and Yan Liu. Benchmarking deep learning models on large healthcare datasets. *Journal of biomedical informatics*, 83:112–134, 2018.

[84] Yongming Qu and Ilya Lipkovich. Propensity score estimation with missing values using a multiple imputation missingness pattern (MIMP) approach. *Statistics in medicine*, 28(9):1402–1414, 2009.

[85] Mahmudur Md Rahman and Sanjay Purushotham. Pseudonam: A pseudo value based interpretable neural additive model for survival analysis. In *AAAI 2021 Fall Symposium on Human Partnership with Medical AI: Design, Operationalization, and Ethics (AAAI-HUMAN 2021)*, 2021.

[86] Kan Ren, Jiarui Qin, Lei Zheng, Zhengyu Yang, Weinan Zhang, Lin Qiu, and Yong Yu. Deep recurrent survival analysis. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4798–4805, 2019.

[87] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[88] David Rindt, Robert Hu, David Steinsaltz, and Dino Sejdinovic. Survival regression with proper scoring rules and monotonic neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 1190–1205. PMLR, 2022.

[89] Donald B Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976.

[90] Joseph L Schafer and John W Graham. Missing data: our view of the state of the art. *Psychological methods*, 7(2):147, 2002.

[91] Fabian Scheipl. spikeslabgam: Bayesian variable selection, model choice and regularization for generalized additive mixed models in r. *Journal of statistical software*, 43:1–24, 2011.

[92] Matthieu Scherpf, Felix Gräßer, Hagen Malberg, and Sebastian Zaunseder. Predicting sepsis with a recurrent neural network using the mimic iii database. *Computers in biology and medicine*, 113:103395, 2019.

[93] Daniel Servén and Charlie Brummitt. pygam: Generalized additive models in python. *Zenodo*, 2018.

[94] Anis Sharafoddini, Joel A Dubin, David M Maslove, Joon Lee, et al. A new insight into missing data in intensive care unit patient profiles: observational study. *JMIR medical informatics*, 7(1):e11605, 2019.

[95] Matthew Sperrin and Glen P Martin. Multiple imputation with missing indicators as proxies for unmeasured variables: simulation study. *BMC medical research methodology*, 20(1):1–11, 2020.

[96] Matthew Sperrin, Glen P Martin, Rose Sisk, and Niels Peek. Missing data should be handled differently for prediction than for description or causal explanation. *Journal of Clinical Epidemiology*, 125:183–187, 2020.

[97] Aude Sportisse, Claire Boyer, and Julie Josse. Estimation and imputation in probabilistic principal component analysis with missing not at random data. *Advances in Neural Information Processing Systems*, 33:7067–7077, 2020.

[98] D Mikis Stasinopoulos and Robert A Rigby. Generalized additive models for location scale and shape (gamlss) in r. *Journal of Statistical Software*, 23:1–46, 2008.

[99] Daniel J Stekhoven and Peter Bühlmann. Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2012.

[100] Agus Sudjianto, Aijun Zhang, Zebin Yang, Yu Su, and Ningzhou Zeng. Piml toolbox for interpretable machine learning model development and diagnostics. *arXiv preprint arXiv:2305.04214*, 2023.

[101] Krithika Suresh, Cameron Severn, and Debashis Ghosh. Survival prediction models: an introduction to discrete-time modeling. *BMC medical research methodology*, 22(1):207, 2022.

[102] Shengpu Tang, Parmida Davarmanesh, Yanmeng Song, Danai Koutra, Michael W Sjoding, and Jenna Wiens. Democratizing ehr analyses with fiddle: a flexible data-driven preprocessing pipeline for structured clinical data. *Journal of the American Medical Informatics Association*, 27(12):1921–1934, 2020.

[103] Terry Therneau, Cindy Crowson, and Elizabeth Atkinson. Using time dependent covariates and time dependent coefficients in the cox model. *Survival Vignettes*, 2(3):1–25, 2017.

[104] Bheki ETH Twala, MC Jones, and David J Hand. Good methods for coping with missing data in decision trees. *Pattern Recognition Letters*, 29(7):950–956, 2008.

[105] Nikolaus Umlauf, Nadja Klein, Thorsten Simon, and Achim Zeileis. bamlss: a lego toolbox for flexible bayesian regression (and beyond). *Journal of Statistical Software*, 100:1–53, 2021.

[106] Lev V Utkin, Egor D Satyukov, and Andrei V Konstantinov. Survnam: The machine learning survival model explanation. *Neural Networks*, 147:81–102, 2022.

[107] Stef Van Buuren. *Flexible imputation of missing data*. CRC press, 2018.

[108] Stef Van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, 45:1–67, 2011.

[109] Guy Van den Broeck, Anton Lykov, Maximilian Schleich, and Dan Suciu. On the tractability of shap explanations. *Journal of Artificial Intelligence Research*, 74:851–886, 2022.

[110] Mike Van Ness, Billy Block, and Madeleine Udell. Dnamite: Interpretable calibrated survival analysis with discretized additive models. *arXiv preprint arXiv:2411.05923*, 2024.

[111] Mike Van Ness, Tomas Bosschieter, Natasha Din, Andrew Ambrosy, Alexander Sandhu, and Madeleine Udell. Interpretable survival analysis for heart failure risk prediction. In *Machine Learning for Health (ML4H)*, pages 574–593. PMLR, 2023.

[112] Mike Van Ness, Tomas M Bosschieter, Roberto Halpin-Gregorio, and Madeleine Udell. The missing indicator method: From low to high dimensions. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5004–5015, 2023.

[113] Mike Van Ness and Madeleine Udell. Interpretable prediction and feature selection for survival analysis. *arXiv preprint arXiv:2404.14689*, 2024.

[114] Mike Van Ness and Madeleine Udell. dnamite: A python package for neural additive models. *arXiv preprint arXiv:2503.07642*, 2025.

[115] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

[116] Feilong Wang, Wenzhi Pan, Shuming Pan, Junbo Ge, Shuyun Wang, and Miao Chen. Red cell distribution width as a novel predictor of mortality in icu patients. *Annals of medicine*, 43(1):40–46, 2011.

[117] Shirly Wang, Matthew BA McDermott, Geeticka Chauhan, Marzyeh Ghassemi, Michael C Hughes, and Tristan Naumann. Mimic-extract: A data extraction, preprocessing, and representation pipeline for mimic-iii. In *Proceedings of the ACM conference on health, inference, and learning*, pages 222–235, 2020.

[118] Lee-Jen Wei. The accelerated failure time model: a useful alternative to the cox regression model in survival analysis. *Statistics in medicine*, 11(14-15):1871–1879, 1992.

[119] Simon N Wood. mgcv: Gams and generalized ridge regression for r. *R news*, 1(2):20–25, 2001.

[120] Katarzyna Woźnica and Przemysław Biecek. Does imputation matter? benchmark for predictive models. *arXiv preprint arXiv:2007.02837*, 2020.

[121] Liangchen Xu and Chonghui Guo. Coxnam: An interpretable deep survival analysis model. *Expert Systems with Applications*, 227:120218, 2023.

[122] Chengrun Yang, Jicong Fan, Ziyang Wu, and Madeleine Udell. Automl pipeline selection: Efficiently navigating the combinatorial space. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1446–1456, 2020.

[123] Zebin Yang, Aijun Zhang, and Agus Sudjianto. Gami-net: An explainable neural network based on generalized additive models with structured interactions. *Pattern Recognition*, 120:108192, 2021.

[124] Jinsung Yoon, James Jordon, and Mihaela Schaar. Gain: Missing data imputation using generative adversarial nets. In *International conference on machine learning*, pages 5689–5698. PMLR, 2018.

[125] Yuxuan Zhao and Madeleine Udell. Matrix completion with quantified uncertainty through low rank gaussian copula. *Advances in Neural Information Processing Systems*, 33:20977–20988, 2020.

[126] Yuxuan Zhao and Madeleine Udell. Missing value imputation for mixed data via gaussian copula. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 636–646, 2020.

[127] Yuxuan Zhao and Madeleine Udell. gcimpute: A package for missing data imputation. *arXiv preprint arXiv:2203.05089*, 2022.

[128] Yibing Zhu, Jin Zhang, Guowei Wang, Renqi Yao, Chao Ren, Ge Chen, Xin Jin, Junyang Guo, Shi Liu, Hua Zheng, et al. Machine learning prediction models for mechanically ventilated patients: analyses of the mimic-iii database. *Frontiers in medicine*, 8:662340, 2021.