An Introduction to Deep Survival Analysis Models for Predicting Time-to-Event Outcomes

George H. Chen[1]

[1]Heinz College of Information Systems and Public Policy, Carnegie Mellon University

September 29, 2024

Abstract

Many applications involve reasoning about time durations before a critical event happens—also called *time-to-event outcomes*. When will a customer cancel a subscription, a coma patient wake up, or a convicted criminal reoffend? Accurate predictions of such time durations could help downstream decision-making tasks. A key challenge is *censoring*: commonly, when we collect training data, we do not get to observe the time-to-event outcome for every data point. For example, a coma patient has not woken up yet, so we do not know the patient's time until awakening. However, these data points should not be excluded from analysis as they could have characteristics that explain why they have yet to or might never experience the event.

Time-to-event outcomes have been studied extensively within the field of *survival analysis* primarily by the statistical, medical, and reliability engineering communities, with textbooks already available in the 1970s and '80s. Recently, the machine learning community has made significant methodological advances in survival analysis that take advantage of the representation learning ability of deep neural networks. At this point, there is a proliferation of deep survival analysis models. How do these models work? Why? What are the overarching principles in how these models are generally developed? How are different models related?

This monograph aims to provide a reasonably self-contained modern introduction to survival analysis. We focus on predicting time-to-event outcomes at the individual data point level with the help of neural networks. Our goal is to provide the reader with a working understanding of precisely what the basic time-to-event prediction problem is, how it differs from standard regression and classification, and how key "design patterns" have been used time after time to derive new time-to-event prediction models, from classical methods like the Cox proportional hazards model to modern deep learning approaches such as deep kernel Kaplan-Meier estimators and neural ordinary differential equation models. We further delve into two extensions of the basic time-to-event prediction setup: predicting which of several critical events will happen first along with the time until this earliest event happens (the competing risks setting), and predicting time-to-event outcomes given a time series that grows in length over time (the dynamic setting). We conclude with a discussion of a variety of topics such as fairness, causal reasoning, interpretability, and statistical guarantees.

Our monograph comes with an accompanying code repository that implements every model and evaluation metric that we cover in detail: `https://github.com/georgehc/survival-intro`

Contents

1 Introduction

Predicting time durations before a critical event happens arises in numerous applications. These durations are called *time-to-event outcomes*. For example, an e-commerce company may be interested in predicting a user's time until making a purchase (*e.g.*, Chapelle 2014). A video streaming service may be interested in predicting when a customer will stop watching a show (*e.g.*, Hubbard et al. 2021). In healthcare, hospitals may be interested in predicting when a patient's disease will relapse (*e.g.*, Zupan et al. 2000). In criminology, courts may be interested in predicting the time until a convicted criminal reoffends (*e.g.*, Chung et al. 1991). Accurate predictions for these time-to-event outcomes could help in decision-making tasks such as showing targeted advertisements or promotions in the e-commerce or video streaming examples, planning treatments to reduce a patient's risk of disease relapse in the healthcare example, and making bail decisions in the criminology example.

A defining feature of time-to-event prediction problems is that commonly, when we collect training data to learn a model from, we do not get to see the true time-to-event outcome for every data point, *i.e.*, their time-to-event outcome is *censored*. As an example, some training points (*e.g.*, a coma patient) might not have experienced the critical event of interest yet (*e.g.*, waking up). Discarding the points that have not experienced the event would be unwise: they could have characteristics that make them much less likely to experience the event (*e.g.*, the patient's brain activity is highly abnormal).

1.1 Survival Analysis and Time-to-Event Outcomes: Some History and Commentary on Naming

Time-to-event outcomes have been studied for hundreds of years if not longer, where the initial focus was on predicting time until death. Early analyses introduced the use of "life tables", which in a nutshell contain counts such as numbers of births and deaths over time. Graunt [1662] published what might be the first life table and looked at the chance of survival for different age groups.[1] This particular dataset from London was challenging since the survival times (age at the time of death) were not actually recorded, corresponding to a censoring problem. Instead, Graunt largely guessed survival times based on causes of death, which were recorded (albeit they were not necessarily accurate). A few decades later, Halley [1693] analyzed a life table collected from modern day Wrocław and computed the probability of dying within the next year. Halley used these probabilities to determine how to price an annuity (roughly, an expected payout over a person's remaining lifetime). For a historical account of life tables and more generally reasoning about survival times, see for instance the book by Bacaër [2011] and the bibliographical notes accompanying the different chapters of Namboodiri and Suchindran [2013]—these readings altogether walk through highlights from hundreds of years of research on survival times leading up to modern time.

So much of the pioneering research on time-to-event outcomes was on time until death that the enterprise of modeling time-to-event outcomes is now commonly called *survival analysis*, with textbooks already available decades ago (*e.g.*, Mann et al. 1974, Kalbfleisch and Prentice 1980, Cox and Oakes 1984, Fleming and Harrington 1991). Countless other (text)books on survival analysis have since been written and have mainly originated from statistical, medical, and reliability engineering communities (*e.g.*, Klein and Moeschberger 2003, Machin et al. 2006, Selvin 2008, Kleinbaum and Klein 2012, Li and Ma 2013, Harrell 2015, Klein et al. 2016, Ebeling 2019, Prentice and Zhao 2019, Gerds and Kattan 2021, Collett 2023), and at this point, there is also a book tailored to social scientists [Box-Steffensmeier and Jones, 2004].

We want to emphasize though that as the examples we opened the monograph with showed, the critical event need not be death, meaning that we might not be reasoning about "survival" literally. In fact, some researchers work on survival analysis but in titling their papers choose to opt for more general phrasing such as "time-to-event modeling" (*e.g.*, Chapfuwa et al. 2018). We further

[1] As noted by Glass [1963] and Bacaër [2011] among others, there has been some debate as to whether Graunt or his friend William Petty wrote the book but regardless, Graunt's book had five editions published between 1662 and 1676 (for which our citation just uses the earliest year).

emphasize that the "time" in "time-to-event outcome" does not literally have to measure time. For example, a survival analysis model could be used to predict how many units of an inventory item (*e.g.*, a newspaper) to stock the next day given past days' sales counts, so that the "time-to-event outcome" here measures an integer number of items [Huh et al., 2011]. Ultimately, "survival" analysis or "time-to-event" models have been broadly applied to numerous applications far beyond reasoning about either "survival" or "time-to-event" outcomes in a literal sense.

1.2 Machine Learning Models for Survival Analysis

The phrase "machine learning" was only coined in 1959 [Samuel, 1959], the year after the highly influential paper by Kaplan and Meier [1958] came out that analyzed a survival model based on life tables using what is called the "product-limit" estimator [Böhmer, 1912]. (Kaplan and Meier's estimator remains one of the major workhorses of modern time-to-event data analysis; we will see it and deep learning versions of it later in this monograph.) Suffice it to say, machine learning as a field is young compared to survival analysis. Precisely when the first machine learning survival analysis model came about is perhaps not entirely straightforward to trace, in part because nowadays, what is considered a "machine learning model" depends on who one asks. While we may consider k nearest neighbor and kernel survival analysis [Beran, 1981] and survival trees [Ciampi et al., 1981, Gordon and Olshen, 1985] to be machine learning models, would the authors of these original papers?

Fast-forwarding to present time, there is now an explosion in the number of machine learning survival analysis models available. For much larger lists of models than what we cover in this monograph, see the excellent surveys by Wang et al. [2019] and Wiegrebe et al. [2023]. As part of their survey, Wiegrebe *et al.* provide an online catalog of over 60 deep-learning-based survival models (which we will just abbreviate throughout this monograph as *deep survival models*).[2] This catalog is not exhaustive!

With so many machine learning models for survival analysis, what exactly are the major innovations? When and why do different models work? How do they relate to each other? What are overarching patterns in model development? In answering these questions, we think that it is extremely important to distinguish between innovations that are specific to time-to-event prediction vs ones that are not. For the purposes of this monograph, we want to focus on the former as they could help us better understand what is special about time-to-event prediction that helps us build better models.

1.3 The Motivation for This Monograph

We set out to write this monograph for two key reasons:

- First, we wanted to provide a reasonably self-contained introductory text that covers the key concepts of survival analysis with a focus on time-to-event prediction *at the individual data point level* and that also exploits the availability of now standard neural network software. We focus on neural network survival models (*i.e.*, deep survival models) because these models are easy to modify (*e.g.*, to accommodate different data modalities, add loss terms, set a custom learning rate schedule, *etc*). Note that every model that we present in detail has publicly available source code (we discuss software shortly in Section 1.6.4). For readers who are new to survival analysis but are already very comfortable working with standard neural network software at the level of writing custom models and loss functions, we hope that our monograph provides enough survival analysis background to make implementing deep survival models from "scratch" using standard neural network software fairly straightforward.
- Second, we wanted to clearly convey how several major categories of deep survival models are related, and how in deriving these different survival models, we use some of the same key design patterns or derivation techniques over and over again. We hope that by leading the reader through many examples, these patterns will become apparent.

[2] `https://survival-org.github.io/DL4Survival/`

To the best of our knowledge, no existing text provides the sort of introduction to survival analysis that our monograph aims to be. The surveys of machine learning survival models [Wang et al., 2019, Wiegrebe et al., 2023] are not written nor intended for the purpose of giving the reader a working knowledge of how to actually derive survival models from first principles. Meanwhile, the vast majority of survival analysis (text)books do not cover neural networks or deep learning due to how new these are (an example of a textbook that covers neural networks for survival analysis can be found in Chapter 11 of Dybowski and Gant [2001], but this book pre-dates the invention of nearly all the deep survival models we cover).

Overall, we hope that our introduction to survival analysis provides the reader with a solid understanding of what precisely the time-to-event problem setup is, why it is different from standard regression and classification, and how to build survival models with the help of neural networks. We also hope that the reader learns a little bit about where the state-of-the-art is in terms of a variety of other topics that we mention but do not discuss in detail, such as how fairness, causal reasoning, and interpretability play into survival models, and what progress has been made on theoretically analyzing some of these models.

1.4 Monograph Overview and Outline

Our coverage is not meant to remotely be exhaustive in showcasing how deep survival models have been used for time-to-event prediction. We specifically cover the following:

- **Basic Time-to-Event Prediction Setup (Section 2).** We first go over the standard time-to-event prediction problem setup. We state its statistical framework, its prediction task, common ways of writing a likelihood function to be maximized (maximum likelihood is the standard way of learning time-to-event prediction models), and how to evaluate prediction accuracy. Along the way, we lead the reader through various example models to help solidify concepts, all of which could be related to maximizing likelihood functions: exponential and Weibull time-to-event prediction models, DeepHit [Lee et al., 2018], Nnet-survival [Gensheimer and Narasimhan, 2019], the Kaplan-Meier estimator [Kaplan and Meier, 1958], and the Nelson-Aalen estimator [Nelson, 1969, Aalen, 1978]. Importantly, we distinguish between modeling time as continuous vs discrete since the math involved is a bit different. This section also discusses how time-to-event prediction relates to classification and regression.

- **Deep Proportional Hazards Models (Section 3).** We next cover perhaps the most widely used family of time-to-event prediction models in practice, which are called *proportional hazards models*. We define proportional hazards models in a general manner in terms of neural networks. Special cases include the exponential and Weibull models from Section 2, the classical Cox model [Cox, 1972], and DeepSurv [Faraggi and Simon, 1995, Katzman et al., 2018]. Proportional hazards models make a strong assumption that, in some sense, decouples how time contributes to a prediction and how a data point's features contribute to a prediction. This assumption often does not hold in practice. We present a generalization of the DeepSurv model called Cox-Time [Kvamme et al., 2019] that removes this proportional hazards assumption.

- **Deep Conditional Kaplan-Meier Estimators (Section 4).** One of the standard models we encounter in Section 2 is the Kaplan-Meier estimator, which is extremely popular in practice and also different from deep proportional hazards models because it is *nonparametric* (*i.e.*, it does not assume the time-to-event outcome's distribution has a parametric form). However, it only works to describe a population and does not provide predictions for individual data points. We present deep learning versions of the Kaplan-Meier estimator that can make predictions at the individual level. Namely, we cover deep kernel survival analysis [Chen, 2020] and its generalization called survival kernets [Chen, 2024]; the latter can scale to large datasets, can in some sense be interpreted in terms of clusters, and has a statistical guarantee on accuracy for a special case of the model.

- **Neural Ordinary Differential Equation Formulation of Time-to-Event Prediction (Section 5).** We then present a model that can encode all the models we presented in preceding sections, where we phrase the standard time-to-event prediction problem in terms of a *neural ordinary differential equation* model. We specifically go over the neural ODE time-to-event prediction model by Tang et al. [2022b] called SODEN. In presenting SODEN, we also mention some model classes that we did not previously point out, such as deep accelerated failure time models and deep extended hazard models [Zhong et al., 2021].

- **Beyond the Basic Time-to-Event Prediction Setup: Multiple Critical Events and Time Series as Raw Inputs (Section 6).** Whereas all the previous sections used the basic time-to-event prediction setup from Section 2, we now consider two generalizations. First we consider the so-called *competing risks setting* where there are multiple critical events of interest (*e.g.*, for a coma patient, we consider the patient waking up and the patient dying as two different critical events; note that censoring could still happen but is not considered as one of the critical events). We aim to predict *which critical event will happen first* and also the *time until this earliest critical event happens*. The example model we use here is DeepHit [Lee et al., 2018]. Note that the special case of there being one critical event reduces the problem to the one from Section 2. We then generalize the competing risks setting further by considering what happens when we want to make predictions as we see more and more of a given a time series (the dynamic setting). The example model we use here is Dynamic-DeepHit [Lee et al., 2019].

- **Discussion (Section 7).** We end the monograph by discussing a variety of topics that we either only briefly glossed over or that we did not mention at all. For example, we discuss different kinds of censoring, ways to encourage a survival model to be "fair", causal reasoning with survival models, interpretability of deep survival models, issues of statistical guarantees, and more.

Specifically for the example models we cover in Sections 2 to 5, we show how these models relate in Figure 1. When one model is a child of another in this figure, it means that the child model could be represented (possibly with a known approximation) by the parent model. Note that in the figure, just because two models do not overlap does *not* mean that they cannot represent the same underlying true time-to-event outcome distribution. For example, even though deep extended hazard models [Zhong et al., 2021] and survival kernets [Chen, 2024] do not overlap in the figure, they can represent many of the same time-to-event outcome distributions.

We emphasize that just because SODEN [Tang et al., 2022b] can in principle represent all the other models we cover in Sections 2 to 4 (possibly with an approximation), that does not mean that one is best off just using SODEN. An important point is that many of these models are trained in different ways. SODEN's training procedure may not work the best for some of the simpler model classes that it can represent. In particular, it invokes calls to an ordinary differential equation (ODE) solver, which could be overkill if we just want to use one of the simpler models (that has its own simpler training procedure which typically is faster to run). By relying on an ODE solver, we could also run into numerical stability issues that occasionally arise with ODE solvers.

Separately, it is good to keep in mind that deep survival models that we cover allow the modeler to flexibly choose a "base" neural network to use with the model. For example, when working with tabular data, the modeler could choose the base neural network to be a multilayer perceptron. When working with images, the modeler could choose the base neural network to be a convolutional neural network or a vision transformer. And so forth. Such base neural networks could be chosen to be arbitrarily complicated (*e.g.*, we could use as many layers and as many hidden nodes as we would like). Consequently, many deep survival models could in theory be considered equally expressive in what sorts of time-to-event outcomes they can model. However, in practice, training these deep models often requires using standard neural network tricks such as using early stopping, weight decay, dropout, *etc.* Roughly, we would train these models with some regularization to prevent overfitting, and how this regularization impacts different models then depends on their different modeling assumptions.

We remark that the example models we chose to present in this monograph are not necessarily

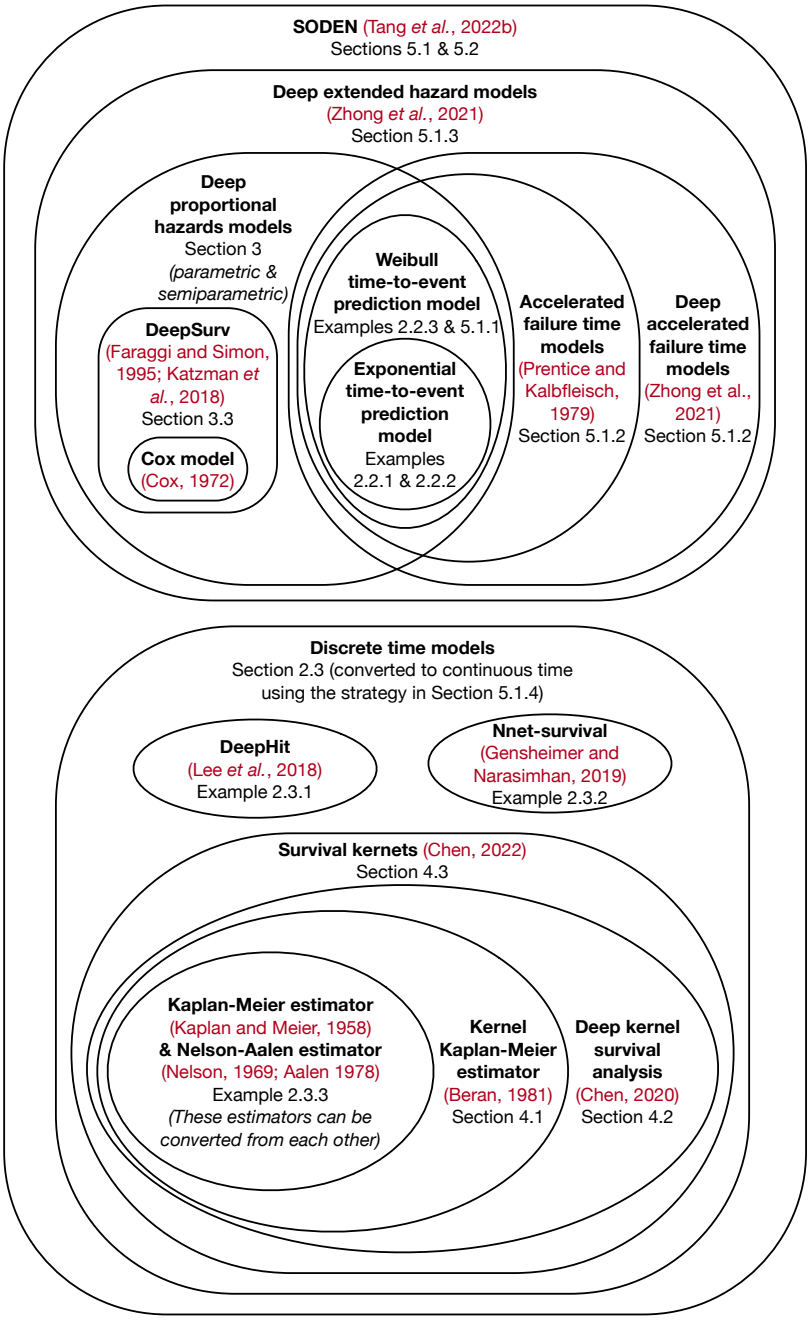


Figure 1: An overview of the models we cover in detail in Sections 2 to 5. One model being the child of another means that the child model could be represented (possibly with a known approximation) by the parent model. Note that when interpreting this diagram, two non-overlapping models could still possibly represent the same underlying time-to-event outcome distribution. For example, deep extended hazard models [Zhong et al., 2021] and survival kernets [Chen, 2024] are capable of modeling many of the same time-to-event outcome distributions. Note that we also cover Cox-Time [Kvamme et al., 2019], which does not easily fit in the diagram; Cox-Time is a generalization of the semiparametric model called DeepSurv [Faraggi and Simon, 1995, Katzman et al., 2018], but Cox-Time can also represent models that are not deep extended hazard models.

the "best". Instead, they were chosen largely for pedagogical considerations and also to showcase some classes of models that are quite different from one another. There are plenty of other time-to-event prediction models (deep-learning-based or not) that work well! By understanding the fundamental concepts in our monograph, the reader should be well-equipped to understand many of these other models. As a reminder, the surveys by Wang et al. [2019] and Wiegrebe et al. [2023] provide fairly extensive listings of many existing machine learning models for time-to-event prediction.

1.5 Examples of Topics Beyond the Scope of Our Monograph

To elaborate a bit more on our scope of coverage, our monograph focuses on survival models for *prediction* that are estimated via *maximum likelihood estimation* in a *neural network framework*. We acknowledge that many survival analysis methods were originally derived for the purpose of statistical inference (*e.g.*, reasoning about population-level quantities and constructing confidence intervals for these quantities) rather than for prediction, including the classical Kaplan-Meier estimator [Kaplan and Meier, 1958] and the Cox model [Cox, 1972]. Our emphasis in this monograph, however, is on learning survival models for prediction, as this is what neural survival models currently are well-suited for. As such, we typically will not cover how to address questions of statistical inference. For readers interested in learning more about statistical inference with survival models, we point out that the various (text)books mentioned in Section 1.1 cover statistical inference results for classical models.[3]

Next, many survival models are learned in a manner that is fundamentally not based on maximum likelihood estimation. For example, countdown regression [Avati et al., 2020] defines a score function to optimize that does not correspond to the usual survival likelihood used in the literature. Meanwhile, Chapfuwa et al. [2018] train a survival model using adversarial learning (with a generative adversarial network [Goodfellow et al., 2014]) rather than maximum likelihood. As another example, random survival forests [Ishwaran et al., 2008] are trained in a greedy manner, where one cannot easily write down a global objective function that is being optimized.

In fact, many decision tree survival models are not optimized in a neural network framework at all, such as the aforementioned random survival forests [Ishwaran et al., 2008] as well as XGBoost [Chen and Guestrin, 2016] (note that the official implementation of XGBoost supports survival analysis), optimal survival trees [Bertsimas et al., 2022], or optimal sparse survival trees [Zhang et al., 2024]. While it is possible to set up learning a decision tree survival model in a neural network framework [Sun and Qiu, 2023], at the time of writing, this line of research appears to still be in early development.

For ease of exposition, we do not cover latent variable models for survival analysis (*e.g.*, Nagpal et al. 2021a,b, Manduchi et al. 2022, Moon et al. 2022, Chen et al. 2024). These particular models build on the ideas we present in this monograph and further use tools not covered in this monograph, notably that of variational inference (*e.g.*, Blei et al. 2017). We think that a reader who understands the fundamentals of our monograph and of variational inference should be well-versed in understanding latent variable models for survival analysis.

1.6 Preliminaries

Before we move onto other sections, we go over some prerequisite knowledge that we will assume that the reader is familiar with. We then explain how we view neural networks and the notation that we use throughout the rest of the monograph. At the end of this section, we provide links to some available software packages and to our companion code repository for this monograph.

[3] As a concrete example, the textbook by Klein and Moeschberger [2003] routinely explains how to construct confidence intervals for various estimated quantities, such as confidence intervals for survival functions obtained from the Kaplan-Meier estimator (see Section 4.3 of their book) and the Cox model (see Section 8.8 of their book).

1.6.1 Prerequisites

We assume that the reader knows calculus, introductory probability and statistics, and the basics of machine learning, especially neural networks, including how to code them up and optimize them in standard neural network software (*e.g.*, PyTorch [Paszke et al., 2019], TensorFlow [Abadi et al., 2015], JAX [Bradbury et al., 2018]). For example, we assume that the reader knows how to run minibatch gradient descent using a standard neural network optimizer (*e.g.*, Adam [Kingma and Ba, 2015]). For a primer on neural networks, see, for instance, the interactive textbook *Dive into Deep Learning* by Zhang et al. [2023].[4]

In terms of neural network architectures that the reader should already know to understand our monograph, we have intentionally tried to keep this listing short:

- (Sections 2 to 5 and the first half of Section 6) The reader should know multilayer perceptrons for classification and regression (corresponding to the case where raw input data are fixed-length feature vectors). For example, the reader should know that softmax activation yields a probability distribution, and that the function defined by an inner product $\mathbf{f}(x;\theta) := x^\top\theta$ for $x, \theta \in \mathbb{R}^d$ is a special case of a multilayer perceptron. (Note that we present the material in a general manner where the raw inputs need not be fixed-length feature vectors.)
- (Second half of Section 6) In the latter half of Section 6, in addition to multilayer perceptrons, the reader should also know recurrent neural networks (RNNs). RNNs enable us to work with variable-length time series as raw inputs.[5]

1.6.2 How We View Neural Networks

As we mentioned in Section 1.4, deep survival models that we cover all depend on a base neural network. By analogy, if we were tackling a classification problem with k classes using deep learning, then the standard strategy is to specify a base neural network (such as a multilayer perceptron) and then we feed the output of the base neural network to a linear layer (also called a full-connected layer or a dense layer) with k output nodes and softmax activation (so that the output of the overall network consists of predicted probabilities of the k classes).[6] Then when we learn the network, we use a classification loss function (*e.g.*, cross entropy loss). The final linear layer added with k output nodes and softmax activation is referred to as a "prediction head". If instead of classification, we were looking at a regression problem (predicting a single real number), then we could set the prediction head to be a linear layer with 1 output node and no nonlinear activation.

When working with deep survival models for time-to-event prediction, the idea is the same. We first specify a base neural network. Afterward, to get the overall network to predict a time-to-event outcome, it is as simple as choosing a "survival layer" at the end (to serve as the prediction head) and using an appropriate survival loss. Depending on the survival layer chosen, there are restrictions on what the output of the base neural network is. For example, when we cover the Cox proportional hazards model, we will see that the base neural network should be set to output a single real number (which could be interpreted as a risk score), and there is actually no additional layer to add. The loss function is then specified a particular way for model training using these "risk scores".

Every survival model we cover could be thought of as a different possible survival layer to use as the prediction head. Each model comes with a loss function. For the models we cover, the loss function will always be a negative log likelihood loss with possibly some other loss terms added, depending on the model.

Extremely importantly, we will typically not spell out details of how to set the base neural network aside from what we require of its output, meaning that we usually intentionally leave the specific architecture choices up to the modeler. We do this precisely because standard tricks can be used for how to choose the base neural network (as we mentioned above, we could choose a

[4] `https://D2L.ai`

[5] While we do not explicitly cover nor assume that the reader knows transformers, we point out that transformers can also handle variable-length inputs (so that in our coverage, RNNs can actually be replaced by transformers).

[6] This strategy would require the base neural network to output some number of nodes that should be at least k (if it is less than k, then we would have trouble representing all k classes).

multilayer perceptron when working with tabular data, a convolutional neural network or a vision transformer when working with images, *etc*). This also means that advances in neural network technology that are not specific to time-to-event prediction could also trivially be incorporated. For example, if we were to work with multimodal data such as the raw inputs being both images and text, then we could choose the base neural network to be based off a model such as CLIP [Radford et al., 2021]. *An important implication is that when we cover an existing deep survival model in detail, even if the original authors of the model provided architecture details in their paper, we omit the architecture details that are not essential to understanding the design of their overall model.*

Another reason why we do not state very specific neural network architectures to use is because the technology has rapidly been changing! The latest trends in neural network architectures today might be out of fashion tomorrow. To complicate matters, depending on the dataset used in a time-to-event prediction task, which specific architecture works the best might vary, and also which neural network optimizer we should use and with what learning schedule might also vary. Our monograph does not dwell on these engineering details, which are important in practice but are not needed in understanding the core high-level concepts.

1.6.3 Notation

We typically use uppercase letters (*e.g.*, X) to denote random variables and lowercase letters (*e.g.*, x) to denote deterministic quantities such as constants or specific realized values of random variables. Functions could either be uppercase or lowercase, where we have tried to stick to common conventions used in survival analysis literature (*e.g.*, the so-called conditional survival function is represented by uppercase S). Bold letters (*e.g.*, $\mathbf{f}$) are usually used to represent parametric functions such as neural networks. We also frequently use the notation $[m] := \{1, 2, \ldots, m\}$, where m is a positive integer. When we use the "log" function, we always mean natural log.

Optimization problems regularly appear in the monograph. When we write $\widehat{\theta} := \arg\min_\theta \mathbf{L}(\theta)$, where $\mathbf{L}$ is a loss function with parameter variable θ, this minimization would be carried out using (some variant of) minibatch gradient descent and, technically, we are usually not finding a solution that achieves the global minimum.

1.6.4 Software Packages and Datasets

As our exposition assumes that the reader is familiar with standard neural network software that have developer communities that primarily work in Python, we list some Python survival analysis packages in Table 1. This list is not exhaustive. We list packages for both deep and non-deep survival models since we think that trying both is important in practice. Per package, we list some (not all) of the models and evaluation metrics implemented. We anticipate that over time, many of these packages will add functionality. Overall, the current state of software packages that support deep survival models is a bit scattered: no single package is—in our opinion—sufficiently comprehensive, and at the time of writing, some packages have not been regularly maintained.

Currently, the packages in Table 1 do not implement all the models that we cover in detail. SODEN [Tang et al., 2022b], deep kernel survival analysis [Chen, 2020], survival kernets [Chen, 2024], and Dynamic-DeepHit [Lee et al., 2019] are not currently included in the software packages in Table 1, but their code is available from the original authors; see the links in Table 2.

In terms of publicly available survival datasets, the `pycox` software package comes with datasets that are all sufficiently large for learning neural network models (mostly in the thousands of data points along with one dataset with roughly 3 million points). The `scikit-survival` and `lifelines` packages also come with datasets; some are a bit small though (a few hundred or fewer points).

Table 1: Some software packages used for survival analysis/time-to-event prediction. Models in blue and evaluation metrics in red are ones that we cover in detail in this monograph.

Package	Link	Some supported methods (not exhaustive)
`scikit-survival` [Pölsterl, 2020]	`https://github.com/sebp/scikit-survival`	Kaplan-Meier estimator[1], Nelson-Aalen estimator[2], Cox model[3], various survival tree ensemble methods including random survival forests[4], concordance index[5], time-dependent concordance index (truncated)[6], time-dependent AUC[7], Brier score[8]
`lifelines` [Davidson-Pilon, 2019]	`https://github.com/CamDavidsonPilon/lifelines`	Kaplan-Meier estimator[1], Nelson-Aalen estimator[2], Cox model[3] and regularized variants, accelerated failure time (AFT) models[9], concordance index[5]
`xgboost` [Chen and Guestrin, 2016]	`https://github.com/dmlc/xgboost`	XGBoost supports using Cox and accelerated failure time loss functions
`glmnet_python` [Simon et al., 2011]	`https://github.com/bbalasub1/glmnet_python`	Cox model[3] and regularized variants; this is the official port of `glmnet` from R
`pycox` [Kvamme et al., 2019]	`https://github.com/havakv/pycox`	unified PyTorch implementations of DeepSurv[10], Cox-Time[11], Nnet-survival[12], DeepHit[13], N-MTLR[14], time-dependent concordance index (not truncated)[15], Brier score[8]
`pysurvival` [Fotso et al., 2019]	`https://github.com/square/pysurvival`	N-MTLR implementation by original author[14], random survival forests[4]
`auton-survival` [Nagpal et al., 2022b]	`https://github.com/autonlab/auton-survival`	DeepSurv[10], Deep Survival Machines[16], Deep Cox Mixtures[17]
`SurvivalEVAL` [Qi et al., 2024a]	`https://github.com/shi-ang/SurvivalEVAL`	concordance index[5], Brier score[8], D-calibration[18], margin[18] and pseudo-observation[19] MAE scores
`torchsurv` [Monod et al., 2024]	`https://github.com/Novartis/torchsurv`	Cox model[3], Weibull AFT model[9], concordance index[5], time-dependent AUC[7], Brier score[8]

[1]Kaplan and Meier [1958] [2]Nelson [1969], Aalen [1978] [3]Cox [1972]
[4]Ishwaran et al. [2008] [5]Harrell et al. [1982] [6]Uno et al. [2011]
[7]Uno et al. [2007], Hung and Chiang [2010] [8]Graf et al. [1999] [9]Prentice and Kalbfleisch [1979]
[10]Faraggi and Simon [1995], Katzman et al. [2018] [11]Kvamme et al. [2019]
[12]Gensheimer and Narasimhan [2019] [13]Lee et al. [2018] [14]Fotso [2018]
[15]Antolini et al. [2005] [16]Nagpal et al. [2021a] [17]Nagpal et al. [2021b]
[18]Haider et al. [2020] [19]Qi et al. [2023]

Table 2: Some models that we cover that are not currently implemented in the packages in Table 1.

Model	Link
Deep kernel survival analysis [Chen, 2020]	`https://github.com/georgehc/dksa`
Survival kernets [Chen, 2024]	`https://github.com/georgehc/survival-kernets`
SODEN [Tang et al., 2022b]	`https://github.com/jiaqima/SODEN`
Dynamic-DeepHit [Lee et al., 2019]	`https://github.com/chl8856/Dynamic-DeepHit`

Companion code repository. To help readers with starting to work with deep survival analysis models in Python, we provide Python code that accompanies our monograph in the following code repository:

`https://github.com/georgehc/survival-intro`

This repository includes sample code for every model and every evaluation metric that we discuss in detail. Our code shows how to train different deep survival models, use them to predict time-to-event outcomes, and evaluate the quality of the predictions using some standard evaluation metrics. Our code is primarily in the form of Jupyter notebooks, which include a mix of code cells and explanations for different parts of the code. As we progress through the monograph, we point to specific Jupyter notebooks in our code repository for readers interested in seeing how concepts we cover get translated into code.

Our code has been written with pedagogy in mind. We stick to using standard PyTorch conventions, and we have written our notebooks at a level that exposes the main neural net optimization loop (minibatch gradient descent) and highlights where base neural networks appear in various deep survival models. Our code aims to make various preprocessing and model training steps more transparent, so that if one wants to modify any part of these, doing so should be straightforward.

Moreover, for ease of exposition, our notebooks that accompany Sections 2 through 5 all use the same standard dataset SUPPORT [Knaus et al., 1995], for which we predict the time until death of

severely ill hospitalized patients with various diseases.[7] Our notebooks that accompany Section 6 use the PBC dataset [Fleming and Harrington, 1991], which is on predicting the time until death and the time until transplantation of patients with primary biliary cirrhosis of the liver; here, death and transplantation are viewed as competing events where we only observe whichever one happens first for a training patient (or alternatively, if neither has happened for a training patient, then we at least know the last check-up time with the patient).

Importantly, in our Jupyter notebooks, we do *not* extensively optimize hyperparameters for any particular deep survival model to try to push the prediction performance of the model to be as good as possible. Thus, the final evaluation scores obtained in our notebooks should not be interpreted as the best possible scores achievable by the different models we implement. Furthermore, our code is not written to be "production-grade" with, for instance, extensive sanity checks or unit tests.

Lastly, we anticipate occasionally updating our code notebooks to accommodate updates to software packages, to improve exposition or clarity, or to fix bugs that are discovered. The latest version will be available at the GitHub link provided above.

2 Basic Time-to-Event Prediction Setup

We now describe the standard problem setup in time-to-event prediction. We use this problem setup for much of the rest of the monograph. Our goal in this section is to give a reasonably self-contained introduction to the math involved for time-to-event prediction. By the end of this section, the reader should have an understanding of what the standard problem setup is, how prediction tasks are defined, what the general strategy is for learning many time-to-event prediction models (maximum likelihood), how to measure prediction accuracy, and how the time-to-event prediction setup relates to the more mainstream prediction tasks in machine learning of classification and regression.

This section is organized as follows:

- (Section 2.1) We first go over the statistical framework for the standard time-to-event prediction problem setup.
- (Section 2.2) We then go over time-to-event prediction when modeling time as continuous. Note that we separate our coverage of continuous vs discrete time as the math involved is a bit different. Beginning by modeling time as continuous, we go over common prediction tasks that correspond to estimating a few different "target" functions. By relating the statistical framework from Section 2.1 to these target functions, we can state a general likelihood function. This likelihood function is important because most time-to-event prediction models we know of can be stated as maximizing a likelihood function, possibly accounting for additional regularization or loss terms, or constraints. Along the way, we provide a couple illustrative examples of simple parametric time-to-event prediction models (special cases of what are called proportional hazards models [Cox, 1972] and accelerated failure time models [Prentice and Kalbfleisch, 1979]).
- (Section 2.3) Turning to modeling time as discrete, we again go over prediction tasks in terms of target functions, followed by going over the standard likelihood function used. The example models we showcase are DeepHit [Lee et al., 2018], Nnet-survival [Gensheimer and Narasimhan, 2019], the Kaplan-Meier estimator [Kaplan and Meier, 1958], and the Nelson-Aalen estimator [Nelson, 1969, Aalen, 1978]. In the discrete time setting (unlike in continuous time), the likelihood function relates to classification at different discretized time points.
- (Section 2.5) Next, we discuss some approaches to evaluating the quality of predictions. Specifically, we go over ranking-based accuracy metrics, various mean absolute error and mean squared error metrics, and a calibration metric.

[7] For these particular code notebooks, we also provide an example of how to modify the code to work with different data, with the concrete example being training on the Rotterdam tumor bank dataset [Foekens et al., 2000] and then testing on the German Breast Cancer Study Group dataset [Schumacher et al., 1994]; these two datasets are on predicting survival times of breast cancer patients.

- (Section 2.6) We provide additional commentary on how the standard time-to-event prediction setup relates to classification and regression. Notably, there is an approach called *survival stacking* [Craig et al., 2021] that enables one to use existing probabilistic binary classifiers for time-to-event prediction, but this reduction comes at a potentially steep computational cost.

We occasionally mention technicalities with details that we defer to Section 2.A. Understanding these details is not essential for understanding the rest of the monograph.

For ease of exposition, we phrase terminology in terms of time until death. Of course, as we already pointed out in Section 1, the critical event of interest in real applications need not be death.

2.1 Standard Right-Censored Statistical Framework

We assume that we have n training points $(X_1, Y_1, \Delta_1), \ldots, (X_n, Y_n, \Delta_n)$, where training point $i \in [n]$ has raw input $X_i \in \mathcal{X}$ (such as a fixed-length feature vector, an image, a text document, *etc*), observed time $Y_i \in [0, \infty)$, and "event indicator" $\Delta_i \in \{0, 1\}$: if $\Delta_i = 1$ (the critical event happened for the i-th point), then Y_i is the true survival time; otherwise, Y_i is the "censoring" time (which could be thought of as the last time we checked on the i-th point, when it was still alive). Classically, each data point corresponds to a different person.

We use X to denote the random variable corresponding to a generic raw input, T to denote the random variable corresponding to the true (possibly unobserved) survival time corresponding to raw input X, and C to denote the random variable corresponding to the true (possibly unobserved) censoring time corresponding to raw input X. For these three random variables, we assume that there are the following three probability distributions that are unknown:

- $\mathbb{P}_X$ is the probability distribution for random raw input X, where we assume that the support of this distribution is the raw input space $\mathcal{X}$ (we formally define the support in Section 2.A.1). In this monograph, we take $\mathcal{X}$ to be any input space that standard neural network software can work with.
- $\mathbb{P}_{T|X}(\cdot|x)$ is the conditional probability distribution of survival time T given $X = x$.
- $\mathbb{P}_{C|X}(\cdot|x)$ is the conditional probability distribution of censoring time C given $X = x$.

The training points $(X_1, Y_1, \Delta_1), \ldots, (X_n, Y_n, \Delta_n)$ are assumed to be independent and identically distributed (i.i.d.), where each point (X_i, Y_i, Δ_i) for $i \in [n]$ is generated as follows:

1. Sample raw input X_i from $\mathbb{P}_X$.
2. Sample true survival time T_i from $\mathbb{P}_{T|X}(\cdot|X_i)$.
3. Sample true censoring time C_i from $\mathbb{P}_{C|X}(\cdot|X_i)$.
4. If $T_i \leq C_i$ (death happens before censoring): output $Y_i = T_i$ and $\Delta_i = 1$ (no censoring).

   Otherwise: output $Y_i = C_i$ and $\Delta_i = 0$ (the true survival time is censored).

Note that conditioned on X_i, the two random variables T_i and C_i are independent (this assumption is commonly referred to as "independent censoring"). Moreover, even though T_i and C_i show up in the generative procedure, we do not observe both of them. Instead, we observe exactly one of them. The one we observe is stored in the variable Y_i, and the variable Δ_i tells us whether we observed the survival time (when $\Delta_i = 1$) or the censoring time (when $\Delta_i = 0$) for the i-th training point.

We point out that step 4 of the generative procedure above could equivalently be written in a more concise manner: we set the observed time to be $Y_i = \min\{T_i, C_i\}$ and the event indicator to be $\Delta_i = \mathbb{1}\{T_i \leq C_i\}$, where $\mathbb{1}\{\cdot\}$ is the indicator function which is equal to 1 if its argument is true and is equal to 0 otherwise. Notationally, for a generic random raw input X with true survival time T and true censoring time C, we denote the observed time as $Y = \min\{T, C\}$ and the event indicator $\Delta = \mathbb{1}\{T \leq C\}$. In other words, each training point (X_i, Y_i, Δ_i) is i.i.d. with the same distribution as (X, Y, Δ).

Technically, the statistical framework that we have described is referred to as *right-censored*, which just means that for the censored data (the data for which $\Delta_i = 0$), the true survival time is *after* the observed censoring time. We discuss other types of censored data in Section 7.1 (namely, where the survival time is some time *before* the observed censoring time or, separately, where the survival time is known to be within an interval).

Remark 2.1 (Defining time 0)**.** Extremely importantly, we have to be precise what we mean by time 0 across the training data (sometimes, this time is referred to as the "time of origin"). Put another way, for the i-th point, exactly what time is Y_i measured starting from? For example, for a video streaming service that wants to predict the time until a customer stops watching a show, time 0 could be defined to mean when each the customer first starts streaming the show. Thus, for different customers, their time 0 could correspond to different actual world times. In a healthcare example, if we are looking at coma patients in an intensive care unit (ICU) and we want to predict the time until they awaken, then we may want to define each patient's time 0 to mean when they were first admitted to the ICU. In general, time 0 is typically defined to correspond to what is called a "synchronization event" (in the two aforementioned examples, this synchronization event would be a user starting to stream a show, and a patient getting admitted to the ICU).

2.2 Time-to-Event Prediction in Continuous Time

Modeling time as continuous, we formally define a few common time-to-event prediction tasks, which could be thought of as estimating specific *prediction target* functions (Section 2.2.1). Using the statistical framework from Section 2.1, we can then derive a likelihood function that we can maximize to learn a time-to-event prediction model (Section 2.2.2).

Key assumptions. For test raw input $x \in \mathcal{X}$, we assume that the survival time T conditioned on $X = x$ is a continuous random variable with probability density function (PDF) $f(t|x)$ and a cumulative distribution function (CDF) $F(t|x) = \int_0^t f(u|x)\mathrm{d}u$; either of these functions fully characterizes the distribution $\mathbb{P}_{T|X}(\cdot|x)$.

2.2.1 Prediction Targets

Survival function. The first prediction target (also called an *estimand* in statistics terminology) that we present is called the *conditional survival function*, given by

$$\begin{aligned} S(t|x) &:= \mathbb{P}(\text{survive beyond time } t \mid \text{raw input is } x) \\ &= \mathbb{P}(T > t|X = x) \\ &= 1 - \mathbb{P}(T \leq t|X = x) \\ &= 1 - F(t|x), \end{aligned} \tag{1}$$

where $t \geq 0$ and $x \in \mathcal{X}$. In this monograph, we refer to the conditional survival function $S(\cdot|x)$ simply as the *survival function* since our notation already indicates that we are conditioning on x. Predicting $S(\cdot|x)$ means estimating an entire function (*i.e.*, a curve)—*not* just a single number (survival time)—for test raw input x. In literature, this function is sometimes called the *survivor function* (*e.g.*, Kalbfleisch and Prentice 1980), the *reliability function* (*e.g.*, Ebeling 2019), or the *complementary CDF* (*e.g.*, Downey 2011).

As equation (1) indicates, the true survival function $S(\cdot|x)$ is 1 minus the CDF $F(\cdot|x)$. A few implications are as follows:

(a) $S(\cdot|x) = 1 - F(\cdot|x)$ monotonically decreases from 1 to 0 since any CDF monotonically increases from 0 to 1.

(b) Estimating the function $S(\cdot|x)$ is equivalent to estimating the CDF $F(\cdot|x)$, which means that we aim to estimate the conditional survival time distribution $\mathbb{P}_{T|X}(\cdot|x)$.

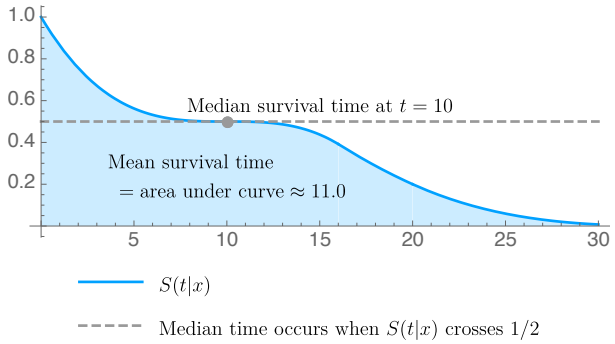


Figure 2: Example of a survival function and its median and mean survival times.

(c) If we want a single number survival time estimate for raw input x, we can back one out if we know (an estimate of) $S(\cdot|x)$. We give two ways of doing this:

- *Median survival time.* Where a CDF crosses $1/2$ corresponds to a median of a distribution, so finding a time t for which $S(t|x) = 1 - F(t|x) = 1/2$ gives a *median* survival time of raw input x. In practice, we only have an estimate $\widehat{S}(\cdot|x)$ of $S(\cdot|x)$ so we find t such that $\widehat{S}(t|x) \approx 1/2$.
- *Mean survival time.* For any nonnegative random variable A, recall that $\mathbb{E}[A] = \int_0^\infty \mathbb{P}(A > u)\mathrm{d}u$. Thus, the mean survival time of raw input x is $\mathbb{E}[T|X = x] = \int_0^\infty \mathbb{P}(T > u|X = x)\mathrm{d}u = \int_0^\infty S(u|x)\mathrm{d}u$, the area under the survival function. In practice, we numerically integrate the survival function estimate $\widehat{S}(\cdot|x)$.

See Figure 2 for an example survival function and its corresponding median and mean survival times.[8]

Different time-to-event prediction models make different assumptions on $S(\cdot|x)$ and often predict transformed variants of $S(\cdot|x)$ rather than predicting $S(\cdot|x)$ directly. The next two prediction targets we discuss are both transformed versions of $S(\cdot|x)$.

Hazard function. A transformed version of $S(\cdot|x)$ that is commonly predicted is the so-called *hazard function*:

$$h(t|x) := -\frac{\mathrm{d}}{\mathrm{d}t}\log S(t|x) = -\frac{\frac{\mathrm{d}}{\mathrm{d}t}S(t|x)}{S(t|x)} = -\frac{\frac{\mathrm{d}}{\mathrm{d}t}[1 - F(t|x)]}{S(t|x)} = \frac{f(t|x)}{S(t|x)}, \tag{2}$$

where, as a reminder, $f(\cdot|x)$ is the PDF of distribution $\mathbb{P}_{T|X}(\cdot|x)$. *The right-most expression of equation (2) says that the hazard function is the instantaneous rate of death conditioned on surviving up to time t for raw input x.* Importantly, for the same reason why PDFs are nonnegative but could otherwise have arbitrarily large positive values, the hazard function is also only nonnegative and could have arbitrarily large positive values. Perhaps the most widely used time-to-event prediction model, the Cox proportional hazards model [Cox, 1972], is stated in terms of the hazard function.

[8] In practice, we may only know or have an estimate of $S(t|x)$ when t is not too large. For instance, we might only know $S(t|x)$ for all $t \in [0, t_{\max}]$ where $t_{\max}$ is some finite time horizon. In this case, it is possible that $S(t_{\max}|x) > 1/2$, so that we would only know that the median survival time is some time after $t_{\max}$. In this scenario, if we would still like a median survival time estimate that is a single number, then we would need some extrapolation strategy (which could mean having some parametric model for what $S(t|x)$ looks like for $t > t_{\max}$). Similarly, if we only know $S(t|x)$ up to some time horizon $t_{\max}$, then we would not be able to exactly compute the integral $\int_0^\infty S(t|x)dt$ to come up with a mean survival time estimate; we would need some way to extrapolate $S(t|x)$ for $t > t_{\max}$ to assist us in computing the integral.

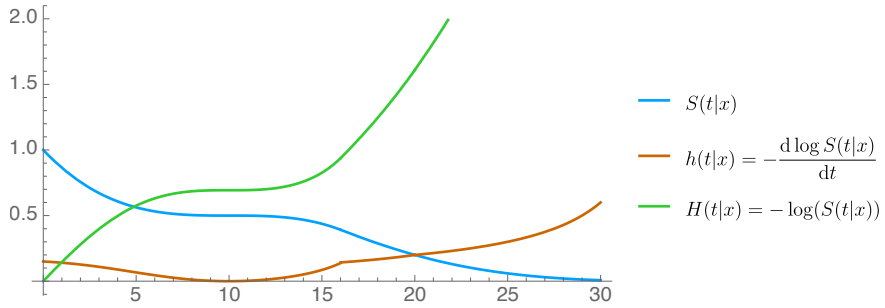


Figure 3: The survival function $S(\cdot|x)$ from Figure 2 along with its hazard $h(\cdot|x)$ and cumulative hazard $H(\cdot|x)$ functions.

If we know $h(\cdot|x)$, then we can recover $S(\cdot|x)$ since

$$h(t|x) = -\frac{\mathrm{d}}{\mathrm{d}t}\log S(t|x) \iff \int_0^t h(u|x)\mathrm{d}u = -\log S(t|x)$$
$$\iff S(t|x) = \exp\Big(-\int_0^t h(u|x)\mathrm{d}u\Big). \tag{3}$$

Cumulative hazard function. Another commonly predicted transformed version of $S(\cdot|x)$ is the *cumulative hazard function*:

$$H(t|x) := \int_0^t h(u|x)\mathrm{d}u. \tag{4}$$

From equation (3), we see that $S(t|x) = \exp(-H(t|x))$, so if we know $H(\cdot|x)$, then we can recover $S(\cdot|x)$. Meanwhile, equation (4) implies that $h(t|x) = \frac{\mathrm{d}}{\mathrm{d}t}H(t|x)$, so if we know $H(\cdot|x)$, then we can recover $h(\cdot|x)$. Some time-to-event prediction models directly estimate the cumulative hazard function such as random survival forests [Ishwaran et al., 2008].

An example survival function $S(\cdot|x)$ and its corresponding hazard function $h(\cdot|x)$ and cumulative hazard function $H(\cdot|x)$ are shown in Figure 3. Note that whereas $S(\cdot|x)$ and $H(\cdot|x)$ are monotonic functions, $h(\cdot|x)$ need not be monotonic.

Because we will often convert between $f(\cdot|x)$, $F(\cdot|x)$, $S(\cdot|x)$, $h(\cdot|x)$, and $H(\cdot|x)$ later in the monograph, we summarize the relationship between these functions below.

Summary 2.1 (Continuous time prediction targets)**.** Suppose that the key assumptions stated at the start of Section 2.2 hold. Let $x \in \mathcal{X}$. The following equations show how the PDF $f(\cdot|x)$, the CDF $F(\cdot|x)$, the survival function $S(\cdot|x)$, the hazard function $h(\cdot|x)$, and the cumulative hazard function $H(\cdot|x)$ are related (where we have time $t \geq 0$):

$$f(t|x) = \frac{\mathrm{d}}{\mathrm{d}t}F(t|x) = \frac{\mathrm{d}}{\mathrm{d}t}(1 - S(t|x)) = h(t|x)S(t|x),$$
$$F(t|x) = \int_0^t f(u|x)\mathrm{d}u = 1 - S(t|x),$$
$$S(t|x) = 1 - F(t|x) = \int_t^\infty f(u|x)\mathrm{d}u = e^{-H(t|x)} = e^{-\int_0^t h(u|x)\mathrm{d}u},$$
$$h(t|x) = \frac{\mathrm{d}H(t|x)}{\mathrm{d}t} = -\frac{\mathrm{d}}{\mathrm{d}t}\log S(t|x) = \frac{f(t|x)}{S(t|x)},$$
$$H(t|x) = -\log S(t|x) = \int_0^t h(u|x)\mathrm{d}u.$$

Importantly, any of these functions fully specifies the conditional survival time distribution

$\mathbb{P}_{T|X}(\cdot|x)$.

2.2.2 Likelihood and Example Models (Parametric Proportional Hazards and Accelerated Failure Time Models)

Now that we have presented the standard right-censored statistical framework and defined some common prediction targets, we state the likelihood function commonly used in deriving many time-to-event prediction models. Specifically, across the n i.i.d. training data, we define the following likelihood that does not depend on the censoring distribution:

$$\mathcal{L} := \prod_{i=1}^{n} \left\{ f(Y_i|X_i)^{\Delta_i} S(Y_i|X_i)^{1-\Delta_i} \right\}. \tag{5}$$

This likelihood could be parsed in an intuitive manner: for the i-th point, if $\Delta_i = 1$ (so the time-to-event outcome is not censored), then the contribution to the product is the PDF of $\mathbb{P}_{T|X}(\cdot|X_i)$ evaluated at the observed time Y_i. Otherwise if $\Delta_i = 0$, the contribution to the product is the probability of seeing a true survival time larger than Y_i given raw input X_i. This latter case crucially uses the observation that, under the statistical framework of Section 2.1, conditioned on $\Delta_i = 0$ and on X_i, the only information we know about the true survival time T_i is that it is after Y_i.

In practice, many time-to-event prediction models are derived by setting one of the conditional functions in Summary 2.1 to have some parametric form. For instance, we could parameterize the hazard function by setting $h(t|x) = \mathbf{h}(t|x;\theta)$ for some user-specified function $\mathbf{h}$ (such as a multilayer perceptron where the final output is a single number constrained to be nonnegative) with parameter θ. In general, the variable θ could consist of multiple parameters, as we see in the following example.

Example 2.1 (Exponential time-to-event prediction model)**.** Suppose that the raw input space is $\mathcal{X} = \mathbb{R}^d$, and we set $h(t|x)$ to be equal to

$$\mathbf{h}(t|x;\theta) := e^{\beta^\top x + \psi} \quad \text{for } t \geq 0, x \in \mathcal{X}, \tag{6}$$

where $\beta \in \mathbb{R}^d$ and $\psi \in \mathbb{R}$ are parameters, and $\beta^\top x = \sum_{j=1}^{d} \beta_j x_j$ is the Euclidean dot product between β and x. Here, θ (which includes all the parameters) would be given by $\theta = (\beta, \psi) \in \mathbb{R}^d \times \mathbb{R}$. In this toy example, the model for the hazard does not depend on the input time t.

We point out that for this toy example, the survival function corresponds to an exponential distribution, which is why we refer to this model as an exponential time-to-event prediction model. In particular, to show why the survival function is for an exponential distribution, we calculate the cumulative hazard function $H(t|x)$ from $h(t|x)$ and then we calculate $S(t|x)$ from $H(t|x)$ (using the conversions from Summary 2.1):

$$H(t|x) = \int_0^t \mathbf{h}(u|x;\theta)\mathrm{d}u = \int_0^t e^{\beta^\top x + \psi}\mathrm{d}u = te^{\beta^\top x + \psi}, \tag{7}$$

$$S(t|x) = \exp(-H(t|x)) = \exp(-te^{\beta^\top x + \psi}). \tag{8}$$

Here, $S(t|x)$ corresponds to an exponential distribution with rate parameter $e^{\beta^\top x + \psi}$.

Before we can plug in a parametric form of $h(t|x)$ (such as equation (6)) into the likelihood function (equation (5)), we first rewrite equation (5) in terms of the hazard function. Recall from Summary 2.1 that: (i) $f(t|x) = h(t|x)S(t|x)$), and (ii) $S(t|x) = e^{-\int_0^t h(u|x)\mathrm{d}u}$. Using (i) and (ii), we

rewrite equation (5) as

$$\begin{aligned}\mathcal{L} &= \prod_{i=1}^{n} \{f(Y_i|X_i)^{\Delta_i} S(Y_i|X_i)^{1-\Delta_i}\} \\ &\overset{\text{(i)}}{=} \prod_{i=1}^{n} \{h(Y_i|X_i)^{\Delta_i} S(Y_i|X_i)\} \\ &\overset{\text{(ii)}}{=} \prod_{i=1}^{n} \Big\{ h(Y_i|X_i)^{\Delta_i} \exp\Big(-\int_0^{Y_i} h(u|X_i)\mathrm{d}u \Big) \Big\}. \end{aligned} \tag{9}$$

Next, we plug in $h(t|x) = \mathbf{h}(t|x;\theta)$ to get the following likelihood function (that we now emphasize to be a function of θ):

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} \Big\{ \mathbf{h}(Y_i|X_i;\theta)^{\Delta_i} \exp\Big(-\int_0^{Y_i} \mathbf{h}(u|X_i;\theta)\mathrm{d}u \Big) \Big\}.$$

We then estimate θ by solving the maximum likelihood optimization problem $\widehat{\theta} := \arg\max_\theta \mathcal{L}(\theta)$ using, for instance, some variant of gradient ascent. Commonly, the log likelihood is used in the optimization instead, which of course yields the same solution, *i.e.*, $\widehat{\theta} = \arg\max_\theta \log \mathcal{L}(\theta)$, where

$$\begin{aligned}\log \mathcal{L}(\theta) &= \log \prod_{i=1}^{n} \Big\{ \mathbf{h}(Y_i|X_i;\theta)^{\Delta_i} \exp\Big(-\int_0^{Y_i} \mathbf{h}(u|X_i;\theta)\mathrm{d}u \Big) \Big\} \\ &= \sum_{i=1}^{n} \Big\{ \Delta_i \log \mathbf{h}(Y_i|X_i;\theta) - \int_0^{Y_i} \mathbf{h}(u|X_i;\theta)\mathrm{d}u \Big\}. \end{aligned} \tag{10}$$

Especially as our monograph emphasizes deep time-to-event prediction models, we point out that usually when working with standard neural network software, we phrase learning a neural network in terms of minimizing a *loss function*. Specifically, we commonly set the loss function to be the negative log likelihood (NLL), averaged across training data:

$$\begin{aligned}\mathbf{L}_{\text{Hazard-NLL}}(\theta) &:= -\frac{1}{n} \log \mathcal{L}(\theta) \\ &= -\frac{1}{n} \sum_{i=1}^{n} \Big\{ \Delta_i \log \mathbf{h}(Y_i|X_i;\theta) - \int_0^{Y_i} \mathbf{h}(u|X_i;\theta)\mathrm{d}u \Big\}. \end{aligned} \tag{11}$$

The averaging helps normalize the resulting loss function's values as we vary the number of training points n.[9] We use a standard neural network optimizer to minimize this loss in minibatches.

Example 2.2 (Exponential time-to-event prediction model, continued)**.** Continuing with Example 2.1, where $\mathcal{X} = \mathbb{R}^d$, $\theta = (\beta, \psi) \in \mathbb{R}^d \times \mathbb{R}$, and $\mathbf{h}(t|x;\theta) = e^{\beta^\top x + \psi}$, we plug this parametric form of $\mathbf{h}(t|x;\theta)$ into equation (11) to get

$$\begin{aligned}\mathbf{L}_{\text{Hazard-NLL}}(\beta,\psi) &= -\frac{1}{n} \sum_{i=1}^{n} \Big\{ \Delta_i(\beta^\top X_i + \psi) - \int_0^{Y_i} e^{\beta^\top X_i + \psi}\mathrm{d}u \Big\} \\ &= -\frac{1}{n} \sum_{i=1}^{n} \{\Delta_i(\beta^\top X_i + \psi) - Y_i e^{\beta^\top X_i + \psi}\}. \end{aligned}$$

We then numerically minimize the loss $\mathbf{L}_{\text{Hazard-NLL}}(\beta,\psi)$ with respect to β and ψ using a

[9] For example, commonly, the training data are split into minibatches for minibatch gradient descent, where we can tune how large these minibatches are (*i.e.*, the batch size). We can look at how the loss function values change as we increase the number of optimization steps and as we vary the batch size. Normalizing helps make sure that the loss function values are comparable across different batch sizes.

neural network optimizer:

$$(\widehat{\beta}, \widehat{\psi}) := \arg \min_{(\beta,\psi)\in\mathbb{R}^d\times\mathbb{R}} \mathbf{L}_{\text{Hazard-NLL}}(\beta, \psi).$$

For any test raw input $x \in \mathcal{X}$, we can predict the survival, hazard, or cumulative functions by just plugging in the estimates $\widehat{\beta}$ and $\widehat{\psi}$ into equations (8), (6), and (7) respectively.

Our companion code repository includes a Jupyter notebook that implements this exponential time-to-event prediction model applied to the SUPPORT dataset [Knaus et al., 1995].[a] Note that this is the first Jupyter notebook of the monograph and includes an explanation of the basic experimental setup used for all of our Jupyter notebooks from Sections 2 to 5. The notebooks accompanying Section 6 also largely build off this first Jupyter notebook. Thus, for the reader interested in learning how to code with deep survival models, we highly recommend going over this first Jupyter notebook in detail prior to looking at the later notebooks. The overall structure of the notebooks is the same: we load and preprocess data, we learn a survival model using training data (typically using minibatch gradient descent), we predict survival functions of test data, and finally we compute evaluation metrics (discussed later in Section 2.5) on test data.

We also provide a modified version of the first notebook that, instead of using the SUPPORT dataset, trains on the Rotterdam tumor bank dataset [Foekens et al., 2000] and tests on the German Breast Cancer Study Group dataset [Schumacher et al., 1994].[b] This second notebook aims to show what code changes are needed to work with different data but is otherwise the same as the first notebook. The rest of the code notebooks for Sections 2 to 5 use the SUPPORT dataset.

[a] `https://github.com/georgehc/survival-intro/blob/main/S2.2.2_Exponential.ipynb`
[b] `https://github.com/georgehc/survival-intro/blob/main/S2.2.2_Exponential_RotterdamGBSG.ipynb`

Example 2.3 (Weibull time-to-event prediction model)**.** Suppose that $\mathcal{X} = \mathbb{R}^d$, and we set $h(t|x)$ to be

$$\mathbf{h}(t|x;\theta) := t^{e^\phi-1}e^{(\beta^\top x)e^\phi+\psi+\phi} \quad \text{for } t \geq 0, x \in \mathcal{X}. \tag{12}$$

This is a generalization of the exponential time-to-event prediction model (from Examples 2.1 and 2.2), which corresponds to the special case where $\phi = 0$. In this more general case, the collection of parameters is $\theta = (\beta, \psi, \phi) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$, and the survival function corresponds to a Weibull distribution. To see this, we calculate the cumulative hazard and survival functions:

$$\begin{aligned} H(t|x) &= \int_0^t \mathbf{h}(u|x;\theta)\mathrm{d}u \\ &= \int_0^t t^{e^\phi-1}e^{(\beta^\top x)e^\phi+\psi+\phi}\mathrm{d}u = t^{e^\phi}e^{(\beta^\top x)e^\phi+\psi}, \qquad (13) \\ S(t|x) &= \exp(-H(t|x)) = \exp(-t^{e^\phi}e^{(\beta^\top x)e^\phi+\psi}) \\ &= \exp\left(-\left(\frac{t}{\exp(-\beta^\top x - \psi e^{-\phi})}\right)^{e^\phi}\right). \qquad (14) \end{aligned}$$

Here, $S(t|x)$ corresponds to a Weibull distribution with shape parameter e^ϕ and scale parameter $\exp(-\beta^\top x - \psi e^{-\phi})$. In particular, $S(t|x)$ is of the form $\exp\big(-(\frac{t}{\text{scale}})^{\text{shape}}\big)$. Note that we use the definition of the Weibull scale parameter found in standard software packages such as SciPy [Virtanen et al., 2020] and R [R Core Team, 2021, "stats" package]. Some texts define the Weibull scale parameter differently (*e.g.*, Collett 2023) although typically the Weibull shape parameter is defined the same way.[a]

In terms of learning model parameters, we can use a neural network optimizer to minimize the loss from equation (11), which in this case is equal to

$$\begin{aligned}
&\mathbf{L}_{\text{Hazard-NLL}}(\beta, \psi, \phi) \\
&= -\frac{1}{n}\sum_{i=1}^{n}\{\Delta_i \log(Y_i^{e^\phi - 1} e^{(\beta^\top X_i)e^\phi + \psi + \phi}) - (Y_i)^{e^\phi} e^{(\beta^\top X_i)e^\phi + \psi}\} \\
&= -\frac{1}{n}\sum_{i=1}^{n}\{\Delta_i[(e^\phi - 1)\log Y_i + (\beta^\top X_i)e^\phi + \psi + \phi] \\
&\qquad\qquad - (Y_i)^{e^\phi} e^{(\beta^\top X_i)e^\phi + \psi}\}.
\end{aligned}$$

Similar to how we proceeded in Example 2.2, we would obtain the estimates $(\widehat{\beta}, \widehat{\psi}, \widehat{\phi}) := \arg\min_{(\beta,\psi,\phi)\in\mathbb{R}^d\times\mathbb{R}\times\mathbb{R}} \mathbf{L}_{\text{Hazard-NLL}}(\beta, \psi, \phi)$, and plug the estimates $\widehat{\beta}$, $\widehat{\psi}$, $\widehat{\phi}$ into equations (14), (12), and (13) to predict survival, hazard, and cumulative hazard functions. We provide a Jupyter notebook that implements this Weibull time-to-event prediction model.[b]

[a] As a technical remark, we point out that our exposition of the Weibull model is not standard compared to what is in existing literature (see, for instance, Section 12.2 of Klein and Moeschberger [2003]) in that we have intentionally stated the model so that the parameters β, ψ, and ϕ are unconstrained. We use this unconstrained parameterization because commonly neural network optimizers treat parameters as unconstrained. Standard tricks are used to constrain parameters. For example, to constrain a scalar parameter $\lambda \in \mathbb{R}$ to be nonnegative, we could define $\lambda = \exp(\varphi)$, treating $\varphi \in \mathbb{R}$ to be an unconstrained parameter that we optimize over (instead of optimizing over λ). More generally, we could set $\lambda = g(\varphi)$, where $g : \mathbb{R} \to [0, \infty)$ is any activation function that outputs a nonnegative value (such as softplus or ReLU). To constrain a parameter to be between 0 and 1, a sigmoid activation could be used. To constrain a collection of parameters to form a probability distribution, the softmax activation function could be used. Etc.

[b] `https://github.com/georgehc/survival-intro/blob/main/S2.2.2_Weibull.ipynb`

The exponential and Weibull time-to-event prediction models are special cases of what are called *proportional hazards models*, the main topic of Section 3. As a preview, proportional hazards models assume that the hazard function has the factorization

$$h(t|x) = \mathbf{h}_0(t;\theta)e^{\mathbf{f}(x;\theta)} \quad \text{for } t \geq 0, x \in \mathcal{X},$$

for some functions (*e.g.*, neural networks) $\mathbf{h}_0(\cdot;\theta) : [0,\infty) \to [0,\infty)$ and $\mathbf{f}(\cdot;\theta) : \mathcal{X} \to \mathbb{R}$ with parameter variable θ. This factorization makes it clear that time t and raw input x contribute to different multiplicative factors of $h(t|x)$. Regardless of what x is, $h(\cdot|x)$ must be proportional to $\mathbf{h}_0(\cdot;\theta)$. Meanwhile, $\mathbf{f}(x;\theta) \in \mathbb{R}$ could be interpreted as a "risk score" for raw input x. When $\mathbf{f}(x;\theta)$ is larger, then this corresponds to the hazard $h(t|x)$ being larger, which in turn corresponds to x tending to have a shorter survival time.

In the exponential time-to-event prediction model, we have $\mathcal{X} = \mathbb{R}^d$, $\mathbf{h}_0(t;\theta) = e^\psi$ and $\mathbf{f}(x;\theta) = \beta^\top x$, where $\theta = (\beta, \psi) \in \mathbb{R}^d \times \mathbb{R}$. In the Weibull time-to-event prediction model, we have $\mathcal{X} = \mathbb{R}^d$, $\mathbf{h}_0(t;\theta) = t^{e^\phi - 1}e^{\psi+\phi}$ and $\mathbf{f}(x;\theta) = (\beta^\top x)e^\phi$, where $\theta = (\beta, \psi, \phi) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$.

As it turns out, the exponential and Weibull time-to-event prediction models are also special cases of what are called *accelerated failure time* (AFT) models, which we discuss more in Section 5.1.2. In a nutshell, AFT models assume that the survival function $S(\cdot|x)$ has the same shape across different raw inputs x except that for different x, this basic shape of the survival function could be stretched along the time axis as to either accelerate or decelerate when death is likely to occur. The neural ordinary differential equation model in Section 5 encompasses both deep proportional hazards and deep AFT models.[10]

[10] This monograph does not cover classical (*i.e.*, non-neural-network-based) AFT models in much detail, largely because the deep survival models that we cover do not require the reader to know results regarding classical AFT models. However, we would like to mention that classical AFT models are very commonly used (for example, the survival analysis losses supported by XGBoost [Chen and Guestrin, 2016] include proportional hazards and AFT losses). For the reader interested in learning more about classical AFT models that do not use neural networks, please see, for instance, Chapter 12 of the textbook by Klein and Moeschberger [2003] or Chapter 3 of the textbook by Box-Steffensmeier and Jones [2004].

2.3 Time-to-Event Prediction in Discrete Time

A key challenge of working in continuous time is that computing the (log) likelihood requires evaluating integrals. Earlier, when we parameterized the continuous time likelihood in terms of the hazard function $\mathbf{h}(\cdot|x;\theta)$, the likelihood involved terms of the form $\int_0^{Y_i} \mathbf{h}(u|X_i;\theta)\mathrm{d}u$. For the exponential and Weibull models, this integral could be evaluated in closed-form. However, to model time-to-event outcomes in as flexible of a manner as possible, there would in general not be a closed-form expression. To this end, many time-to-event prediction models discretize time into a finite grid, converting integrals into easier-to-compute finite sums. Of course, some time-to-event prediction problems are stated in discrete time to begin with (*e.g.*, Huh et al. 2011).

In this section, we state the discrete time versions of the survival, hazard, and cumulative hazard functions (Section 2.3.1). These discrete time versions do not behave quite the same way as their continuous time analogues. In the case where time actually is continuous but we are discretizing it, we explain some common discretization approaches and also point out some interpolation strategies (Section 2.3.2). Afterward, we state the standard discrete time likelihood function used in practice, provide some example time-to-event prediction models, and relate the likelihood function to classification (Section 2.4).

Key assumptions. Suppose that we discretize time into a user-specified grid of L time points $\tau_{(1)}, \tau_{(2)}, \ldots, \tau_{(L)} \in [0, \infty)$ such that $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$ (we point out how time can be discretized in Section 2.3.2). We assume that all training Y_i values have been discretized to take on values among $\tau_{(1)}, \ldots, \tau_{(L)}$. In terms of notation, we use uppercase L since as we point out later, in practice, the maximum number of time steps could be chosen in a way that depends on the training data (which we view as random), in which case L would be a random variable.

2.3.1 Prediction Targets

We denote the survival, hazard, and cumulative hazard functions as $S[\cdot|x]$, $h[\cdot|x]$, and $H[\cdot|x]$ respectively (throughout the monograph, we use functions with square brackets to indicate that time is discrete). The CDF of distribution $\mathbb{P}_{T|X}(\cdot|x)$ is denoted as $F[\cdot|x]$ and its corresponding probability mass function (PMF) is denoted as $f[\cdot|x]$. Namely,

$$f[\ell|x] := \mathbb{P}(T = \tau_{(\ell)}|X = x) \quad \text{for } \ell \in [L],$$

and

$$F[\ell|x] := \mathbb{P}(T \leq \tau_{(\ell)}|X = x) = \sum_{m=1}^{\ell} f[m|x].$$

Since $f[\cdot|x]$ is a PMF, this means that $f[\ell|x] \geq 0$ for each $\ell \in [L]$, and $\sum_{\ell=1}^{L} f[\ell|x] = 1$. Moreover, we have $f[\ell|x] = F[\ell|x] - F[\ell-1|x]$ with the convention that $F[0|x] := 0$.

Survival function. For $x \in \mathcal{X}$, we define the discrete time survival function evaluated at time index $\ell \in [L]$ to be

$$S[\ell|x] := \mathbb{P}(T > \tau_{(\ell)}|X = x) = 1 - F[\ell|x] = 1 - \sum_{m=1}^{\ell} f[m|x]. \tag{15}$$

Thus, if we know either $F[\cdot|x]$ or $f[\cdot|x]$, then we can readily compute $S[\cdot|x]$ using the above equation. Meanwhile, if we know $S[\cdot|x]$, then we can easily compute $F[\cdot|x] = 1 - S[\cdot|x]$.

To compute $f[\cdot|x]$ given $S[\cdot|x]$, note that

$$\begin{aligned} f[\ell|x] &= F[\ell|x] - F[\ell-1|x] \\ &= (1 - S[\ell|x]) - (1 - S[\ell-1|x]) \\ &= S[\ell-1|x] - S[\ell|x] \qquad \text{for } \ell \in [L], \end{aligned} \tag{16}$$

where we use the convention $S[0|x] := 1$.

Hazard function. Next, for time index $\ell \in [L]$, we define the discrete time hazard function $h[\ell|x]$ in a similar manner as the continuous time version in equation (2). Namely, $h[\ell|x]$ is the probability of dying at time $\tau_{(\ell)}$ conditioned on still being alive at time $\tau_{(\ell)}$ for raw input x:

$$\begin{aligned} h[\ell|x] &:= \mathbb{P}(T = \tau_{(\ell)} | X = x, \overbrace{T \geq \tau_{(\ell)}}^{\text{still alive at time } \tau_{(\ell)}}) \\ &= \frac{\mathbb{P}(T = \tau_{(\ell)}, T \geq \tau_{(\ell)} | X = x)}{\mathbb{P}(T > \tau_{(\ell-1)} | X = x)} \\ &= \frac{\mathbb{P}(T = \tau_{(\ell)} | X = x)}{\mathbb{P}(T \geq \tau_{(\ell)} | X = x)} \\ &= \frac{\mathbb{P}(T = \tau_{(\ell)} | X = x)}{\mathbb{P}(T > \tau_{(\ell-1)} | X = x)} \\ &= \frac{f[\ell|x]}{S[\ell-1|x]}. \end{aligned} \tag{17}$$

Importantly, whereas the continuous time version $h(t|x)$ is constrained to be nonnegative and could possibly be larger than 1, in discrete time, $h[\ell|x]$ is a probability so it cannot be larger than 1.

Equation (17) tells us how to compute $h[\cdot|x]$ given both $f[\cdot|x]$ and $S[\cdot|x]$. To compute $h[\cdot|x]$ only using $S[\cdot|x]$, we note that by plugging equation (16) into equation (17), we obtain

$$h[\ell|x] = \frac{S[\ell-1|x] - S[\ell|x]}{S[\ell-1|x]}. \tag{18}$$

This tells us how to convert from $S[\cdot|x]$ to $h[\cdot|x]$.

To convert from $h[\cdot|x]$ to $S[\cdot|x]$, we first derive the following recurrence relation:

$$\begin{aligned} S[\ell|x] &= \mathbb{P}(T > \tau_{(\ell)} | X = x) \\ &= \mathbb{P}(T > \tau_{(\ell-1)} | X = x) \mathbb{P}(T \neq \tau_{(\ell)} | X = x, T > \tau_{(\ell-1)}) \\ &= \underbrace{\mathbb{P}(T > \tau_{(\ell-1)} | X = x)}_{S[\ell-1|x]} \Big[1 - \underbrace{\mathbb{P}(T = \tau_{(\ell)} | X = x, T > \tau_{(\ell-1)})}_{h[\ell|x] \text{ using the initial line of equation (17)}}\Big] \\ &= S[\ell-1|x](1 - h[\ell|x]). \end{aligned}$$

By plugging in $\ell = 1, 2, \ldots,$ we get that:

- $S[1|x] = 1 - h[1|x]$,
- $S[2|x] = (1 - h[1|x])(1 - h[2|x])$,
- $S[3|x] = (1 - h[1|x])(1 - h[2|x])(1 - h[3|x])$,

and so forth. In general, the pattern that emerges is that

$$S[\ell|x] = \prod_{m=1}^{\ell} (1 - h[m|x]) \qquad \text{for } \ell \in [L], \tag{19}$$

which tells us how to convert an estimate of $h[\cdot|x]$ into one of $S[\cdot|x]$.

Equation (19) has the following interpretation: $h[1|x]$ is the probability of dying at time index 1 for raw input x, so surviving beyond time index 1 happens with probability $1 - h[1|x]$. Conditioned on surviving time index 1, then the probability of dying at time index 2 is $h[2|x]$. Thus, the probability of surviving beyond time index 2 conditioned on surviving past time index 1 for raw input x is $1 - h[2|x]$; to get the probability without conditioning, we multiply by the probability of the event we conditioned on: $(1 - h[1|x])(1 - h[2|x])$. The m-th term in the right-hand side of

equation (19) is the probability that we survive beyond time index m conditioned on surviving all previous time indices.

Cumulative hazard function. We define the discrete time cumulative hazard function as

$$H[\ell|x] := \sum_{m=1}^{\ell} h[m|x]. \tag{20}$$

Thus, equation (20) tells us how to convert from $h[\cdot|x]$ to $H[\cdot|x]$. Converting from $H[\cdot|x]$ to $h[\cdot|x]$ can be done in a straightforward manner:

$$h[\ell|x] = H[\ell|x] - H[\ell-1|x] \tag{21}$$

where $H[0|x] := 0$.

However, whereas in continuous time, we had $H(t|x) = -\log S(t|x)$, using the definition of $H[\ell|x]$ in equation (20), it turns out that $H[\ell|x]$ is *not* equal to $-\log S[\ell|x]$.

Proposition 2.1. Let $x \in \mathcal{X}$. Suppose that $h[\ell|x] \in [0,1)$ for all $\ell \in [L]$. Then $H[\cdot|x]$ is a first-order Taylor approximation of $-\log S[\cdot|x]$. In particular, from using a Taylor expansion, we get

$$-\log S[\ell|x] = H[j|x] + \underbrace{\sum_{m=1}^{\ell} \sum_{p=2}^{\infty} \frac{(h[m|x])^p}{p}}_{\substack{\text{second and higher order} \\ \text{Taylor expansion terms}}} \quad \text{for } \ell \in [L].$$

We defer the proof of Proposition 2.1 to Section 2.A.2.

Instead, we can directly relate $H[\cdot|x]$ and $S[\cdot|x]$ in an exact manner by combining equations (20) and (18) to get

$$H[\ell|x] = \sum_{m=1}^{\ell} h[m|x] = \sum_{m=1}^{\ell} \frac{S[m-1|x] - S[m|x]}{S[m-1|x]},$$

which tells us how to convert from $S[\cdot|x]$ to $H[\cdot|x]$. Converting from $H[\cdot|x]$ to $S[\cdot|x]$ can be done in a two-step procedure: first compute $h[\cdot|x]$ based on $H[\cdot|x]$ using equation (21), and then compute $S[\cdot|x]$ based on $h[\cdot|x]$ using equation (19).

We summarize the relationship between $f[\cdot|x]$, $F[\cdot|x]$, $S[\cdot|x]$, $h[\cdot|x]$, and $H[\cdot|x]$ below.

Summary 2.2 (Discrete time prediction targets)**.** Suppose that the key assumptions stated at the start of Section 2.3 hold, where we discretize time into the grid $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$. Let $x \in \mathcal{X}$. The following equations show how the PMF $f[\cdot|x]$, the CDF $F[\cdot|x]$, the survival function $S[\cdot|x]$, the cumulative hazard function $H[\cdot|x]$, and the hazard function $h[\cdot|x]$ are

related (where time index $\ell \in [L]$):

$$\begin{aligned}
f[\ell|x] &= F[\ell|x] - F[\ell-1|x] = S[\ell-1|x] - S[\ell|x] \\
&= h[\ell|x]S[\ell-1|x], \\
F[\ell|x] &= \sum_{m=1}^{\ell} f[m|x] = 1 - S[\ell|x], \\
S[\ell|x] &= 1 - F[\ell|x] = \sum_{m=\ell+1}^{L} f[m|x] = \prod_{m=1}^{\ell}(1 - h[m|x]), \\
h[\ell|x] &= H[\ell|x] - H[\ell-1|x] = \frac{S[\ell-1|x] - S[\ell|x]}{S[\ell-1|x]} \\
&= \frac{f[\ell|x]}{S[\ell-1|x]}, \\
H[\ell|x] &= \sum_{m=1}^{\ell} \frac{S[m-1|x] - S[m|x]}{S[m-1|x]} = \sum_{m=1}^{\ell} h[m|x].
\end{aligned}$$

We use the convention that $F[0|x] = 0$, $S[0|x] = 1$, and $H[0|x] = 0$. Importantly, any of the above functions fully specifies the conditional survival time distribution $\mathbb{P}_{T|X}(\cdot|x)$.

2.3.2 Time Discretization and Interpolation

Time discretization. Assuming that time is not already discretized, then we have to decide on a discretization method. There are many ways to do this. We begin with the classical example used by the Kaplan-Meier estimator [Kaplan and Meier, 1958]: set $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$ to be the unique observed times among the Y_i variables for which death happened. In other words, take the set of times $\{Y_i : i \in [n] \text{ such that } \Delta_i = 1\}$ and sort them (keeping only the unique values) to get $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$.

Importantly, discretizing time can be done with the help of the training data. As another example, we could specify the number of time steps L that we want to use (*e.g.*, 100) and then set $\tau_{(1)}$ to be the smallest (*i.e.*, earliest) time seen among the Y_i variables for which death happened, and then set $\tau_{(L)}$ to be the largest (*i.e.*, latest) time seen among the Y_i variables for which death happened. Then, we could define the rest of the grid points so that they are evenly spaced apart (*i.e.*, $\tau_{(\ell+1)} = \tau_{(\ell)} + \frac{\tau_{(L)} - \tau_{(1)}}{L-1}$ for $\ell \in [L-1]$). Alternatively, we could use even spacing on a log scale (*i.e.*, $\log \tau_{(1)}, \log \tau_{(2)}, \ldots, \log \tau_{(L)}$ are evenly spaced), or even spacing in terms of percentiles (*e.g.*, if $L = 5$, we use the 0%, 25%, 50%, 75%, and 100% percentile values among observed times of death). Of course, the time grid need not be set based on training data if the user has good intuition for how to manually choose it.

As a technical remark, in equation (15), since we stated that $f[\cdot|x]$ is a PMF, this implies that $S(\tau_{(L)}) = 0$. This is not a stringent assumption in that we could easily have set the time grid so that time step $L-1$ is the "last" time step and time step L is a placeholder time step for times that are "too large". For instance, if we discretize time using the Kaplan-Meier approach stated above, we could instead set $L-1$ to be the unique number of times of death (and so $\tau_{(L-1)}$ is the largest time of death encountered in the training data), and we then add a final time step L, where any Y_i value larger than the largest time of death gets discretized to be of the final time step L. During training of many discrete time models, we do not actually need to specify a precise time for the last time step (so that its time could just be considered "$> \tau_{(L-1)}$"), but if for whatever reason an actual time is needed, some maximum time could be specified by the user.

Time interpolation. Sometimes a discrete time model is used, but when making predictions, we may want to switch to using a different time grid from what was used during training (such as a more fine-grain time grid). Naturally, the issue of how to interpolate (or even extrapolate) arises. While a basic strategy like linear interpolation can be used, we point out that Kvamme and Borgan

[2021] discuss more sophisticated interpolation strategies that assume either constant probability density or constant hazard values in between time grid points. Which one works best depends on the dataset at hand.

Importantly, if one wants to use some other interpolation strategy aside from the ones we mentioned already, we suggest checking that the interpolation method appropriately retains the monotonicity of $S(\cdot|x)$ (*i.e.*, the interpolated version should also monotonically decrease). Also, we point out that commonly the earliest time grid point $\tau_{(1)}$ is larger than 0, in which case a standard assumption is that $S(0|x) = 1$ and $S(\cdot|x)$ decays from time 0 to time $\tau_{(1)}$ (how it decays depends on the choice of interpolation method).

2.4 Likelihood, Connection to Classification, and Example Models (DeepHit, Nnet-survival, Kaplan-Meier, Nelson-Aalen)

We now cover the discrete time log likelihood and also relate it to classification. Along the way, we present some example models. We introduce the notation $\kappa(Y_i)$ to denote the specific time index (from $1, 2, \ldots, L$) that time Y_i corresponds to (as a reminder, we assume that we have discretized all Y_i values to the values in $\tau_{(1)}, \ldots, \tau_{(L)}$). Then the likelihood function that does not depend on the censoring distribution is

$$\mathcal{L} := \prod_{i=1}^{n} \{f[\kappa(Y_i)|X_i]^{\Delta_i} S[\kappa(Y_i)|X_i]^{1-\Delta_i}\},$$

which is similar to the continuous time version from equation (5).

Example 2.4 (DeepHit)**.** The DeepHit model [Lee et al., 2018] specifies the survival time PMF $f[\cdot|x]$ in terms of a user-specified neural network $\mathbf{f}(\cdot;\theta) : \mathcal{X} \to [0,1]^L$ with parameter variable θ so that:

$$\begin{bmatrix} f[1|x] \\ f[2|x] \\ \vdots \\ f[L|x] \end{bmatrix} = \begin{bmatrix} \mathbf{f}_1(x;\theta) \\ \mathbf{f}_2(x;\theta) \\ \vdots \\ \mathbf{f}_L(x;\theta) \end{bmatrix} =: \mathbf{f}(x;\theta).$$

Note that the output of $\mathbf{f}(\cdot;\theta)$ needs to be a valid probability distribution (so that $f[\cdot|x]$ is a valid PMF). As an example of how to enforce this, if $\mathbf{f}(\cdot;\theta)$ is a multilayer perceptron, then we could set the last linear layer to output L numbers and have softmax activation.

We could learn θ by maximizing the likelihood function

$$\begin{aligned} \mathcal{L}(\theta) &= \prod_{i=1}^{n} \{f[\kappa(Y_i)|X_i]^{\Delta_i} S[\kappa(Y_i)|X_i]^{1-\Delta_i}\} \\ &= \prod_{i=1}^{n} \left\{ f[\kappa(Y_i)|X_i]^{\Delta_i} \left[\sum_{m=\kappa(Y_i)+1}^{L} f[m|X_i] \right]^{1-\Delta_i} \right\} \\ &= \prod_{i=1}^{n} \left\{ [\mathbf{f}_{\kappa(Y_i)}(X_i;\theta)]^{\Delta_i} \left[\sum_{m=\kappa(Y_i)+1}^{L} \mathbf{f}_m(X_i;\theta) \right]^{1-\Delta_i} \right\}, \end{aligned}$$

where the second equality uses Summary 2.2 (namely that $S[\ell|x] = 1 - F[\ell|x] = 1 - \sum_{m=1}^{\ell} f[m|x] = \sum_{m=\ell+1}^{L} f[m|x]$). In practice, to maximize $\mathcal{L}(\theta)$, we could use a standard neural network optimizer to numerically minimize the negative log likelihood averaged

across training data:

$$\begin{aligned}
&\mathbf{L}_{\text{PMF-NLL}}(\theta) \\
&\quad := -\frac{1}{n}\log \mathcal{L}(\theta) \\
&\quad = -\frac{1}{n}\log \prod_{i=1}^{n}\left\{ [\mathbf{f}_{\kappa(Y_i)}(X_i;\theta)]^{\Delta_i}\left[\sum_{m=\kappa(Y_i)+1}^{L}\mathbf{f}_m(X_i;\theta)\right]^{1-\Delta_i}\right\} \\
&\quad = -\frac{1}{n}\sum_{i=1}^{n}\Bigg\{ \Delta_i \log(\mathbf{f}_{\kappa(Y_i)}(X_i;\theta)) \\
&\qquad\qquad + (1-\Delta_i)\log\left(\sum_{m=\kappa(Y_i)+1}^{L}\mathbf{f}_m(X_i;\theta)\right)\Bigg\}.
\end{aligned}$$

Specifically, we compute

$$\widehat{\theta} := \arg\min_{\theta} \mathbf{L}_{\text{PMF-NLL}}(\theta).$$

After obtaining estimate $\widehat{\theta}$ for θ, we could predict the survival time PMF for any test raw input $x \in \mathcal{X}$ using

$$\widehat{f}[\ell|x] := \mathbf{f}_\ell(x;\widehat{\theta}),$$

from which we could back out estimates of the survival function $S[\ell|x]$, hazard function $h[\ell|x]$, or cumulative hazard function $H[\ell|x]$ using the conversions from Summary 2.2. In particular, we have:

$$\begin{aligned}
\widehat{S}_{\text{DeepHit}}[\ell|x] &= \sum_{m=\ell+1}^{L}\widehat{f}[m|x] = \sum_{m=\ell+1}^{L}\mathbf{f}_m(x;\widehat{\theta}), \\
\widehat{h}_{\text{DeepHit}}[\ell|x] &= \frac{\widehat{f}[\ell|x]}{\widehat{S}[\ell-1|x]} = \frac{\mathbf{f}_\ell(x;\widehat{\theta})}{\sum_{m=\ell}^{L}\mathbf{f}_m(x;\widehat{\theta})}, \\
\widehat{H}_{\text{DeepHit}}[\ell|x] &= \sum_{m=1}^{\ell}\widehat{h}[m|x] = \sum_{m=1}^{\ell}\frac{\mathbf{f}_m(x;\widehat{\theta})}{\sum_{p=m}^{L}\mathbf{f}_p(x;\widehat{\theta})}.
\end{aligned}$$

To see DeepHit in code, please see our accompanying Jupyter notebook.[a] This is the first Jupyter notebook of the monograph that goes over discretizing time prior to learning the model. The same discretization code shows up in a number of our later Jupyter notebooks that involve discrete time survival models.

We remark that the full DeepHit model is more general than the special case of it that we present in this example. In particular, the full DeepHit model adds a second loss term related to ranking (the user has to tune a hyperparameter that trades off between the negative log likelihood loss and the ranking loss) and, furthermore, the full model can keep track of multiple kinds of critical events rather than only a single one such as death (this is referred to as the "competing risks" setup). We present the full DeepHit model in Section 6.1.4 during our coverage of competing risks.

A special case of how to specify the neural network $\mathbf{f}(\cdot;\theta)$ results in a time-to-event prediction model called Multi-Task Logistic Regression [Yu et al., 2011, Fotso, 2018]. Details on this connection are provided by Kvamme and Borgan [2021, Appendix C].

[a] https://github.com/georgehc/survival-intro/blob/main/S2.3.3_DeepHit_single.ipynb

Parameterizing the hazard function and connection to classification. In the continuous case, we showed how we can parameterize the hazard function $h(\cdot|x)$ and minimize a negative log likelihood loss in terms of the hazard function. In discrete time, we could also choose to directly

work with the hazard function $h[\cdot|x]$ (instead of working with the PMF $f[\cdot|x]$ as done by DeepHit). The resulting hazard-function-based likelihood function looks a bit different from that of continuous time (equation (9)) and relates to classification. Using Summary 2.2, we have: (i) $f[\ell|x] = h[\ell|x]S[\ell-1|x]$, and (ii) $S[\ell|x] = \prod_{m=1}^{\ell}(1-h[m|x])$. Then using (i) and (ii), we obtain:

$$\begin{aligned}
\mathcal{L} &= \prod_{i=1}^{n}\{f[\kappa(Y_i)|X_i]^{\Delta_i} S[\kappa(Y_i)|X_i]^{1-\Delta_i}\} \\
&\overset{\text{(i)}}{=} \prod_{i=1}^{n}\{(h[\kappa(Y_i)|X_i]S[\kappa(Y_i)-1|X_i])^{\Delta_i} S[\kappa(Y_i)|X_i]^{1-\Delta_i}\} \\
&\overset{\text{(ii)}}{=} \prod_{i=1}^{n}\left\{\left[h[\kappa(Y_i)|X_i]\prod_{m=1}^{\kappa(Y_i)-1}(1-h[m|X_i])\right]^{\Delta_i}\left[\prod_{m=1}^{\kappa(Y_i)}(1-h[m|X_i])\right]^{1-\Delta_i}\right\} \\
&= \prod_{i=1}^{n}\left\{h[\kappa(Y_i)|X_i]^{\Delta_i}(1-h[\kappa(Y_i)|X_i])^{1-\Delta_i}\left[\prod_{m=1}^{\kappa(Y_i)-1}(1-h[m|X_i])\right]\right\}.
\end{aligned}$$

Take the log of both sides to get

$$\log\mathcal{L} = \sum_{i=1}^{n}\left\{\overbrace{\Delta_i\log(h[\kappa(Y_i)|X_i]) + (1-\Delta_i)\log(1-h[\kappa(Y_i)|X_i])}^{\substack{\text{Bernoulli log likelihood (the negative of the so-called}\\ \text{binary cross entropy loss) at time index } \kappa(Y_i)}} + \underbrace{\sum_{m=1}^{\kappa(Y_i)-1}\log(1-h[m|X_i])}_{\text{death not encountered before time index } \kappa(Y_i)}\right\}. \tag{22}$$

In particular, $h[\cdot|X_i]$ could be viewed as a probabilistic binary classifier for the i-th point where at each time index $\ell \in [L]$, the classifier has a different predicted probability of death conditioned on the i-th point still being alive at time index ℓ. Ideally, $h[\cdot|X_i]$ should be low for all time indices prior to $\kappa(Y_i)$. Then at time index $\kappa(Y_i)$, if death happened, then we want $h[\cdot|X_i]$ to be high; otherwise, we want $h[\cdot|X_i]$ to be low.

Notice that $h[\cdot|X_i]$ could be thought of as a multi-time-horizon classifier: regardless of what time index we make a prediction for, we always condition on the same raw input X_i. In practice, we could think of X_i as information collected prior to time index 1. Using this information, we predict hazard probabilities for all time indices (or time horizons) $1, 2, \ldots, L$.

Example 2.5 (Nnet-survival)**.** The Nnet-survival model [Gensheimer and Narasimhan, 2019] specifies the hazard function $h[\cdot|x]$ in terms of a user-specified neural network $\mathbf{g}(\cdot;\theta) : \mathcal{X} \to \mathbb{R}^L$ with parameter variable θ. In particular,

$$\mathbf{g}(x;\theta) := \begin{bmatrix}\mathbf{g}_1(x;\theta)\\ \mathbf{g}_2(x;\theta)\\ \vdots \\ \mathbf{g}_L(x;\theta)\end{bmatrix},$$

so each output of $\mathbf{g}(\cdot;\theta)$ corresponds to a different time index. Then Nnet-survival sets the hazard function $h[\ell|x]$ equal to

$$\mathbf{h}[\ell|x;\theta] := \frac{1}{1+e^{-\mathbf{g}_\ell(x;\theta)}} \quad \text{for } \ell \in [L], x \in \mathcal{X}, \tag{23}$$

which corresponds to applying the logistic function to each of the L outputs of $\mathbf{g}(\cdot;\theta)$, ensuring that $\mathbf{h}[\ell|x;\theta] \in [0,1]$ for every $\ell \in [L]$.

To learn θ, we maximize the log likelihood in equation (22), which we instead state as minimizing the negative log likelihood loss, averaged across training data:

$$\begin{aligned}\mathbf{L}_{\text{Nnet-survival}}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\Bigg\{&\Delta_i \log(\mathbf{h}[\kappa(Y_i)|X_i;\theta]) \\ &+ (1-\Delta_i)\log(1-\mathbf{h}[\kappa(Y_i)|X_i;\theta]) \\ &+ \sum_{m=1}^{\kappa(Y_i)-1}\log(1-\mathbf{h}[m|X_i;\theta])\Bigg\}. \end{aligned} \tag{24}$$

Plugging equation (23) into equation (24) and using the fact that $1-\mathbf{h}[\ell|x;\theta] = \frac{1}{1+e^{\mathbf{g}_\ell(x;\theta)}}$, we get

$$\begin{aligned}\mathbf{L}_{\text{Nnet-survival}}(\theta) &= \frac{1}{n}\sum_{i=1}^{n}\Bigg\{\Delta_i \log\left(\frac{1}{1+e^{-\mathbf{g}_{\kappa(Y_i)}(X_i;\theta)}}\right) \\ &\qquad + (1-\Delta_i)\log\left(\frac{1}{1+e^{\mathbf{g}_{\kappa(Y_i)}(X_i;\theta)}}\right) \\ &\qquad + \sum_{m=1}^{\kappa(Y_i)-1}\log\left(\frac{1}{1+e^{\mathbf{g}_m(X_i;\theta)}}\right)\Bigg\} \\ &= \frac{1}{n}\sum_{i=1}^{n}\Bigg\{\Delta_i \log(1+e^{-\mathbf{g}_{\kappa(Y_i)}(X_i;\theta)}) \\ &\qquad + (1-\Delta_i)\log(1+e^{\mathbf{g}_{\kappa(Y_i)}(X_i;\theta)}) \\ &\qquad + \sum_{m=1}^{\kappa(Y_i)-1}\log(1+e^{\mathbf{g}_m(X_i;\theta)})\Bigg\}. \end{aligned}$$

Thus, we numerically compute the estimate

$$\widehat{\theta} := \arg\min_{\theta} \mathbf{L}_{\text{Nnet-survival}}(\theta),$$

using a standard neural network optimizer.

Afterward, we can predict the hazard function and from that back out the cumulative hazard and survival functions:

$$\begin{aligned}\widehat{h}_{\text{Nnet-survival}}[\ell|x] &:= \mathbf{h}[\ell|x;\widehat{\theta}] = \frac{1}{1+e^{-\mathbf{g}_\ell(x;\widehat{\theta})}}, \\ \widehat{H}_{\text{Nnet-survival}}[\ell|x] &:= \sum_{m=1}^{\ell}\widehat{h}_{\text{Nnet-survival}}[m|x] = \sum_{m=1}^{\ell}\frac{1}{1+e^{-\mathbf{g}_m(x;\widehat{\theta})}}, \\ \widehat{S}_{\text{Nnet-survival}}[\ell|x] &:= \prod_{m=1}^{\ell}(1-\widehat{h}_{\text{Nnet-survival}}[m|x]) \\ &= \prod_{m=1}^{\ell}\left(1-\frac{1}{1+e^{-\mathbf{g}_m(x;\widehat{\theta})}}\right) = \prod_{m=1}^{\ell}\frac{1}{1+e^{\mathbf{g}_m(x;\widehat{\theta})}}. \end{aligned}$$

As discussed by Kvamme and Borgan [2021, Section 2.2], if the function $\mathbf{g}(\cdot;\theta)$ is a general parametric function that is not restricted to be a neural network, then a number of authors have discussed variants of this same model (*e.g.*, Cox 1972, Brown 1975, Allison 1982, Tutz and Schmid 2016). Our companion code repository includes a Jupyter notebook that covers Nnet-survival.[a]

[a] https://github.com/georgehc/survival-intro/blob/main/S2.3.3_Nnet-survival.ipynb

An alternative way to write the same log likelihood. We now point out an equivalent way of writing the log likelihood equation (22) that is useful for a few reasons: first, this alternative way of writing the log likelihood more clearly shows how the log likelihood could be written as the sum of log likelihood terms from different time indices. In fact, this way of writing the log likelihood leads to a straightforward derivation of a method called the Kaplan-Meier estimator. Second, this alternative way of writing the log likelihood is what is implemented in the now-standard software package `pycox` [Kvamme et al., 2019].

We proceed by defining the matrix $B \in \{0,1\}^{n \times L}$, where the i-th row, ℓ-th column entry is

$$\begin{aligned} B_{i,\ell} &:= \mathbb{1}\{Y_i = \tau_{(\ell)}, \Delta_i = 1\} \\ &= \mathbb{1}\{\kappa(Y_i) = \ell, \Delta_i = 1\} \\ &= \Delta_i \mathbb{1}\{\kappa(Y_i) = \ell\}. \end{aligned}$$

In other words, if the i-th data point's survival time is censored, then the i-th row of B would be all zeros. Otherwise, the i-th row of B would consist of all zeros except for a 1 at column $\kappa(Y_i)$. Note that the matrix B in can readily be computed from $(Y_1, \Delta_1), \ldots, (Y_n, \Delta_n)$.[11]

Then equation (22) can be written as

$$\begin{aligned} \log \mathcal{L} &= \sum_{i=1}^{n} \Bigg\{ \Delta_i \log(h[\kappa(Y_i)|X_i]) + (1-\Delta_i) \log(1 - h[\kappa(Y_i)|X_i]) \\ &\qquad + \sum_{m=1}^{\kappa(Y_i)-1} \log(1 - h[m|X_i]) \Bigg\} \\ &= \sum_{i=1}^{n} \sum_{\ell=1}^{\kappa(Y_i)} \underbrace{\big\{ B_{i,\ell} \log h[\ell|X_i] + (1 - B_{i,\ell}) \log(1 - h[\ell|X_i]) \big\}}_{\text{Bernoulli log likelihood}}, \end{aligned} \tag{25}$$

where we see that every term being added can be thought of as another classification problem. For each data point $i \in [n]$, at each time index $\ell \in [\kappa(Y_i)]$, we check how well the probabilistic classifier $h[\ell|X_i]$ agrees with the data $B_{i,\ell}$.

We can further rearrange equation (25) by exchanging the inner and outer summations to show that each time index can be viewed as having its own loss term:

$$\begin{aligned} &\log \mathcal{L} \\ &= \sum_{i=1}^{n} \sum_{\ell=1}^{\kappa(Y_i)} \big\{ B_{i,\ell} \log h[\ell|X_i] + (1 - B_{i,\ell}) \log(1 - h[\ell|X_i]) \big\} \\ &= \sum_{\ell=1}^{L} \underbrace{\sum_{i=1}^{n} \mathbb{1}\{\ell \le \kappa(Y_i)\} \big\{ B_{i,\ell} \log h[\ell|X_i] + (1 - B_{i,\ell}) \log(1 - h[\ell|X_i]) \big\}}_{\text{likelihood term for time index } \ell}. \end{aligned} \tag{26}$$

This way of writing the log likelihood makes it clear that we could view the problem as doing classification at each of the L time indices (so that it is a multi-time-horizon classification problem).

[11] However, we cannot in general recover $(Y_1, \Delta_1), \ldots, (Y_n, \Delta_n)$ from only knowing the matrix B. To see this, note that if the i-th point is censored, then the i-th row in B would consist of all zeros, from which we cannot recover Y_i.

One could, for example, parameterize $h[\cdot|x]$ so that each time index has its own parameters and also there could be some parameters shared across time.

Example 2.6 (Kaplan-Meier and Nelson-Aalen estimators)**.** Consider a simple setup where for each time index $\ell \in [L]$, we predict the hazard probability to be $\theta_\ell \in [0,1]$ (completely disregarding what the raw input x is). In particular, we use the model

$$h[\ell|x] := \theta_\ell \quad \text{for } \ell \in [L], x \in \mathcal{X}. \tag{27}$$

Because the hazard does not depend on the raw input x, this simple model could be thought of as specifying a population-level discrete time hazard function rather than one that actually accounts for raw inputs. Then the log likelihood in this case, using equation (26) and now emphasizing the dependence on $\theta := (\theta_1, \dots, \theta_L) \in [0,1]^L$, is given by

$$\mathcal{L}(\theta) = \sum_{\ell=1}^{L} \underbrace{\sum_{i=1}^{n} \mathbb{1}\{\ell \le \kappa(Y_i)\}\big\{B_{i,\ell} \log \theta_\ell + (1 - B_{i,\ell}) \log(1-\theta_\ell)\big\}}_{=:\mathcal{L}_{(\ell)}(\theta_\ell) \text{ (likelihood term for time index } \ell)}.$$

Because $\mathcal{L}(\theta) = \sum_{\ell=1}^{L} \mathcal{L}_{(\ell)}(\theta_\ell)$, to maximize $\mathcal{L}(\theta)$ with respect to θ, it suffices to maximize each time index's likelihood term $\mathcal{L}_{(\ell)}(\theta_\ell)$ with respect to θ_ℓ. This maximization has a closed-form solution (we derive this solution in Section 2.A.3):

$$\widehat{\theta}_\ell := \arg\max_{\theta_\ell} \mathcal{L}_{(\ell)}(\theta_\ell) = \frac{D[\ell]}{N[\ell]} \quad \text{for } \ell \in [L], \tag{28}$$

where $D[\ell]$ is the number of deaths that occurred at time index ℓ within the training data, and $N[\ell]$ is the number of points "at risk" (that could possibly die) at time index ℓ within the training data. Formally, recalling that $\tau_{(\ell)}$ is the actual time corresponding to time index $\ell \in [L]$:

$$D[\ell] := \sum_{j=1}^{n} \mathbb{1}\{Y_j = \tau_{(\ell)}\}\Delta_j = \sum_{j=1}^{n} B_{j,\ell}, \tag{29}$$

$$N[\ell] := \sum_{j=1}^{n} \mathbb{1}\{Y_j \ge \tau_{(\ell)}\} = \sum_{j=1}^{n} \mathbb{1}\{\kappa(Y_j) \ge \ell\}. \tag{30}$$

Note that these were classically written out in a table referred to as a "life table". Of course, on a computer, we would typically store $D[\cdot]$ and $N[\cdot]$ each in 1D arrays/"tables".

Using the conversions in Summary 2.2, the estimated hazard, cumulative hazard, and survival functions are (again, these are population-level estimates that do not depend on the test raw input x):

$$\begin{aligned}
\widehat{h}[\ell] &:= \widehat{\theta}_\ell = \frac{D[\ell]}{N[\ell]}, \\
\widehat{H}[\ell] &:= \sum_{m=1}^{\ell} \widehat{h}[m] = \sum_{m=1}^{\ell} \frac{D[m]}{N[m]}, \\
\widehat{S}[\ell] &:= \prod_{m=1}^{\ell} (1 - \widehat{h}[m]) = \prod_{m=1}^{\ell} \Big(1 - \frac{D[m]}{N[m]}\Big).
\end{aligned} \tag{31}$$

In fact, $\widehat{H}[\ell]$ and $\widehat{S}[\ell]$ correspond to the discrete time versions of what are called the Nelson-Aalen estimator [Nelson, 1969, Aalen, 1978] and the Kaplan-Meier estimator [Kaplan and

Meier, 1958], respectively. These estimators are typically stated in continuous time, where the only difference is that we simply interpolate the discrete time estimator so that at any time $t \geq 0$, we output the most recently seen discrete time index's value (also called *forward filling* interpolation).

Specifically, using the convention that $\tau_{(0)} := 0$ and $\widehat{H}_{\text{NA}}[0] := 0$, the Nelson-Aalen estimator is:

$$\begin{aligned}\widehat{H}_{\text{NA}}(t) &:= \begin{cases} \widehat{H}[\ell - 1] & \text{if } \tau_{(\ell-1)} < t \leq \tau_{(\ell)} \text{ for } \ell \in [L], \\ \widehat{H}[L] & \text{if } t > \tau_{(L)}, \end{cases} \\ &= \sum_{m=1}^{L} \mathbb{1}\{\tau_{(m)} \leq t\} \frac{D[m]}{N[m]} \quad \text{for } t \geq 0. \end{aligned} \tag{32}$$

Using the convention that $\tau_{(0)} := 0$ (again) and $\widehat{S}_{\text{KM}}[0] := 1$, the Kaplan-Meier estimator is:

$$\begin{aligned}\widehat{S}_{\text{KM}}(t) &:= \begin{cases} \widehat{S}[\ell - 1] & \text{if } \tau_{(\ell-1)} < t \leq \tau_{(\ell)} \text{ for } \ell \in [L], \\ \widehat{S}[L] & \text{if } t > \tau_{(L)}, \end{cases} \\ &= \prod_{m=1}^{L} \left(1 - \frac{D[m]}{N[m]}\right)^{\mathbb{1}\{\tau_{(m)} \leq t\}} \quad \text{for } t \geq 0. \end{aligned} \tag{33}$$

To summarize, the Nelson-Aalen and Kaplan-Meier estimators are inherently discrete time methods for predicting population-level cumulative hazard and survival functions, and they correspond to using the population-level hazard function (equation (27)) that is estimated using maximum likelihood. We provide a Jupyter notebook that covers both the Nelson-Aalen and Kaplan-Meier estimators, although we mostly focus on the latter.[a]

[a] https://github.com/georgehc/survival-intro/blob/main/S2.3.3_Kaplan-Meier_Nelson-Aalen.ipynb

The Kaplan-Meier and Nelson-Aalen estimators are considered *nonparametric* since they do not impose a parametric form on the survival, cumulative hazard, or hazard functions, and typically they are used in a manner where the time grid is taken to be all unique times of death (so that the parameter variable $\theta \in [0,1]^L$ in Example 2.6 could grow in size since as we collect more training data, the number of unique times of death L could increase).

In fact, under fairly general settings, as the amount of training data grows to ∞, the Kaplan-Meier estimator provably converges to $S_{\text{pop}}(t) := \mathbb{P}(T > t)$ for all times t that are not too large [Földes and Rejtö, 1981] (after all, we cannot expect the Kaplan-Meier estimator to predict survival probabilities well when t is close to or exceeds the maximum observed time of death). This theory could be extended to the Nelson-Aalen estimator as well using the fact that the latter is a first-order Taylor series approximation of the former (using the same argument as in Proposition 2.1).

In practice, the Kaplan-Meier estimator is extremely popular because it can easily be used to compare groups. For example, suppose that we have a group of patients who received a treatment and another group of patients who did not receive the treatment (the control group). We could collect the ground truth labels (the Y_i and Δ_i variables) for all these patients and then compute a Kaplan-Meier survival function just for those who received treatment and, separately, another Kaplan-Meier survival function just for those in the control group. Plotting the two survival functions overlaid over each other could be informative. In fact, there are also statistical tests for comparing the two groups (*e.g.*, testing whether they are the "same") such as the log-rank test [Mantel, 1966]. Of course, we can use this same idea provided that we have any approach for partitioning the complete collection of training data into different groups or clusters and, per cluster, fit a Kaplan-Meier survival function. We will revisit this idea in Section 4.3.

2.5 Evaluation Metrics for Time-to-Event Prediction

The fundamental challenge in measuring accuracy in time-to-event prediction problems is censoring. Consider a training point X_i with observed time Y_i and event indicator $\Delta_i = 0$. This means that Y_i is the censoring time C_i and *not* the true unobserved survival time T_i. Thus, even if a time-to-event prediction model could come up with an estimate $\widehat{T}_i$ of T_i, we would not have a ground truth value to compare $\widehat{T}_i$ with, and there is no guarantee that the observed censoring time Y_i is close to the true unobserved survival time T_i.

We now cover many common evaluation metrics for time-to-event prediction models. Throughout this section, we define evaluation metrics just using the training data, mainly to avoid introducing new notation. Very importantly, these evaluation metrics could also be computed on validation or test data.

We want to emphasize that none of the evaluation metrics we present is perfect for every situation. In fact, many evaluation metrics we present are known to be "improper" [Gneiting and Raftery, 2007], meaning that the best score possible is *not* achieved by the true conditional survival distribution (*i.e.*, an evaluation metric can assign a better score to an incorrect model compared to the correct model). With this cautionary note in mind, we generally recommend using multiple evaluation metrics.

Our Jupyter notebooks that accompany Sections 2 through 5 show how to compute all the evaluation metrics that we discuss in detail in this section. Our code specifically computes these evaluation metrics on held out test data.

2.5.1 Ranking-Based Accuracy Metrics

We begin with accuracy metrics based on ranking. Sometimes, these ranking metrics are referred to as "discrimination" metrics.

Harrell's concordance index. One of the most common accuracy metrics used in survival analysis is Harrell's *concordance index* [Harrell et al., 1982], often abbreviated as "c-index". The c-index is the fraction of pairs of data points that are correctly ranked by a prediction procedure among pairs of data points that can be ranked unambiguously, which as we will see shortly neatly handles censored data. As it is a fraction, c-index values range from 0 to 1, where 1 means the most accurate.

We first work out a simple example before we more formally state the definition of c-index.

Example 2.7 (C-index calculation)**.** Consider if we have three devices:

- Device A fails after 2 days of use.
- Device B fails after 10 days of use.
- Device C is still working after 6 days of use.

In other words, their respective (Y_i, Δ_i) values are $(2, 1)$, $(10, 1)$, and $(6, 0)$. (We take time 0 to be when a device first starts being used. For simplicity, our subsequent exposition is phrased as if we started using the devices at the same time.)

We know for sure that device A failed before device B, and that device A failed before device C. However, we do not know which of device B or C fails first. Thus, in this example, only two pairs (A & B, A & C) consist of data points that can be ranked unambiguously. If a prediction procedure ranks device A as having a shorter survival time than device B, and device A as having a longer survival time than device C, then only one out of the two pairs is correctly predicted, so the c-index is $1/2$.

From this example, we see that computing c-index values requires that our time-to-event prediction model can rank different data points, which is a different task from what we had previously presented in predicting survival, hazard, or cumulative hazard functions of different data points. As we mentioned in Section 2.2.2 and will discuss in detail in Section 3, proportional hazards mod-

els assign a risk score $\mathbf{f}(x;\theta) \in \mathbb{R}$ to each raw input x. Thus, we can rank data points by their risk scores.

We now formally define the c-index in terms of a risk score function (such as the one from a proportional hazards model that is learned using training data).

Definition 2.1 (C-index)**.** Suppose that we have a risk score function $r : \mathcal{X} \to \mathbb{R}$. For any $x, x' \in \mathcal{X}$, if $r(x) > r(x')$, then the risk score predicts x to be "worse off" than x' (*e.g.*, x tends to have a shorter survival time than x').

We first define the set of "comparable pairs" (*i.e.*, pairs of points that could be unambiguously ordered):

$$\mathcal{E} := \{(i,j) \in [n] \times [n] : \Delta_i = 1, Y_i < Y_j\}. \tag{34}$$

Thus, each pair $(i,j) \in \mathcal{E}$ has data point i unambiguously having shorter survival time than data point j since data point i died whereas data point j has a higher observed time (and it does not matter whether data point j died or not). This means that ideally, we should have $r(X_i) > r(X_j)$.

Then we define

$$\text{c-index} := \frac{1}{|\mathcal{E}|} \sum_{(i,j)\in\mathcal{E}} \mathbb{1}\{r(X_i) > r(X_j)\},$$

which is a fraction between 0 and 1. Higher scores are better.

Note that the c-index as defined above aims to estimate

$$\text{c-index}^* := \mathbb{P}(r(X) > r(X') \mid \Delta = 1, Y < Y'),$$

where the two points (X, Y, Δ) and (X', Y', Δ') are i.i.d. samples from the generative procedure in Section 2.1 (the random variable Δ' is not needed though in defining c-index*). We use the superscript "*" to indicate that c-index* is a population-level quantity that depends on the true underlying distributions $\mathbb{P}_X$, $\mathbb{P}_{T|X}$ and $\mathbb{P}_{C|X}$ that we do not know in practice.

Connection to AUC: Similar to the area under the receiver operator characteristic curve (AUC) for binary classification, a c-index score of $1/2$ is considered low and can be achieved via "random guessing". As an example, consider the random risk score function $r_{\text{random}} : \mathcal{X} \to \mathbb{R}$, where $r_{\text{random}}(x)$ just ignores the input x and outputs a random number sampled from a standard Gaussian $\mathcal{N}(0,1)$. One can show that the expected value of the c-index achieved by r_{random} is $1/2$. In fact, when there is no censoring, then the population-level quantity c-index* is equal to the AUC for an appropriately defined classification problem (see, for example, Harrell Jr et al. 1996, Koziol and Jia 2009).

Key limitations: One problem with the c-index metric is that it requires having a risk score function to rank points with. Many time-to-event prediction models do not explicitly learn such a function. One workaround is simple: as we mentioned in Section 2.2.1, given any estimated survival function $\widehat{S}(\cdot|x)$, we could compute a median (or mean) survival time estimate of x, which would enable us to rank points based on their estimated median (or mean) survival times. This workaround is somewhat unsatisfying, as we convert each point's predicted survival function into a single number, and we otherwise ignore the shape of the function.

Another problem with the c-index metric is that it is known to be improper for predicting the risk of a data point dying within a pre-specified time horizon (*e.g.*, 10 years). For details on this result, see the paper by Blanche et al. [2019].

Handling ties in observed times: As a minor technical remark, we point out that the c-index calculation is sometimes defined slightly differently to address what happens if there are two points i and j (with $i \neq j$) that have the same observed time $Y_i = Y_j$ and at least one of them has died. For ease of exposition, we do not include such pairs of points in the set $\mathcal{E}$.

Time-dependent concordance index. Antolini et al. [2005] proposed a time-dependent concordance index (denoted as C^{td}) that aims to make better use of any predicted survival function $\widehat{S}(\cdot|x)$. We state how to compute it before providing intuition for its definition.

Definition 2.2 (C^{td} index)**.** Suppose that we have a survival function estimate $\widehat{S}(\cdot|x)$ for any $x \in \mathcal{X}$. Then using the set of comparable pairs $\mathcal{E}$ from equation (34), we define the C^{td} index as

$$C^{\text{td}} := \frac{1}{|\mathcal{E}|} \sum_{(i,j) \in \mathcal{E}} \mathbb{1}\{\widehat{S}(Y_i|X_i) < \widehat{S}(Y_i|X_j)\},$$

which again is between 0 and 1. Higher scores are better.

To motivate this definition, consider two points $(i, j) \in \mathcal{E}$, so point i died and $Y_i < Y_j$. Then at the earlier time Y_i, we would like point i to be predicted as having *higher* risk of death (in other words, *lower* survival probability) than point j. We can use survival probability $\widehat{S}(Y_i|\cdot)$ as a ranking function in this case that is specific to time Y_i, where $\widehat{S}(Y_i|x)$ being lower for x means that x is predicted to be at *higher* risk of death at time Y_i. Thus, for $(i, j) \in \mathcal{E}$, we would like $\widehat{S}(Y_i|X_i) < \widehat{S}(Y_i|X_j)$.

Importantly, in the case of proportional hazards models (which we previewed at the end of Section 2.2.2 and cover in detail in Section 3), the C^{td} index is the same as the c-index defined earlier, so that the C^{td} index could be viewed as a generalization of the c-index to accommodate any time-to-event prediction model that predicts $\widehat{S}(\cdot|x)$.

Lastly, we point out two theoretical properties. First, the C^{td} score is an empirical estimate of the population-level quantity

$$C^{\text{td}\,*} := \mathbb{P}(\widehat{S}(Y|X) < \widehat{S}(Y|X') \mid \Delta = 1, Y < Y'),$$

where just as before, we assume that the two points (X, Y, Δ) and (X', Y', Δ') are i.i.d. samples from the generative procedure in Section 2.1. Second, we point out that the C^{td} score is known to be improper [Rindt et al., 2022].

Truncated time-dependent concordance index. The c-index and C^{td}-index scores each give only a single number and do not provide an accuracy score at a specific user-specified time $t \in [0, \infty)$. Uno et al. [2011] presented a "truncated" version of the time-dependent concordance index that does depend on t. We begin by defining the time-dependent set of comparable pairs:

$$\mathcal{E}(t) := \{(i, j) \in [n] \times [n] : \Delta_i = 1, Y_i < t, Y_j > Y_i\}.$$

The only change compared to $\mathcal{E}$ is that $\mathcal{E}(t)$ has the additional constraint that $Y_i < t$. Now for $(i, j) \in \mathcal{E}(t)$, we would like $\widehat{S}(t|X_i) < \widehat{S}(t|X_j)$. Then Uno *et al.* defined the following evaluation metric (following Tang et al. [2022b], we denote this metric as C_t^{td}).

Definition 2.3 (Truncated time-dependent concordance index)**.** Let $t \geq 0$. Suppose that we have a survival function estimate $\widehat{S}(\cdot|x)$ for any $x \in \mathcal{X}$. Then using the time-dependent set of comparable pairs $\mathcal{E}(t)$, we define the truncated time-dependent concordance index at time t as

$$C_t^{\text{td}} := \frac{\sum_{(i,j) \in \mathcal{E}(t)} w_i \mathbb{1}\{\widehat{S}(t|X_i) < \widehat{S}(t|X_j)\}}{\sum_{(i,j) \in \mathcal{E}(t)} w_i},$$

where $w_1, w_2, \ldots, w_n \in [0, \infty)$ are so-called *inverse probability of censoring weights* to be defined in a moment (equation (35)). Values of C_t^{td} are between 0 and 1, where higher is better.

If the weights $w_1, w_2, \ldots, w_n$ are all set to be 1, then we would have a score function that looks quite similar to the C^{td} index except now looking at a specific time t. By having weights be unequal, we are changing how much each pair $(i, j) \in \mathcal{E}(t)$ contributes to both the numerator and the denominator of C_t^{td}.

We state how Uno *et al.* set the weights before providing intuition for why. In what follows, we assume that censoring time C is independent of raw input X, meaning that the

conditional censoring distribution $\mathbb{P}_{C|X}(\cdot|x)$ is equal to a population-level censoring distribution $\mathbb{P}_C(\cdot)$. Define $S_{\text{censor}}(t) := \mathbb{P}(C > t)$ for $t \geq 0$. To estimate this function, the standard approach is to fit the Kaplan-Meier estimator (equation (33)) on the training labels $(Y_1, \mathbb{1}\{\Delta_1 = 0\}), (Y_2, \mathbb{1}\{\Delta_2 = 0\}), \ldots, (Y_n, \mathbb{1}\{\Delta_n = 0\})$ (by switching censoring to be the critical event of interest, we reason about time until censoring instead of time until death); the resulting estimated Kaplan-Meier "survival" function is denoted $\widehat{S}_{\text{censor}}(\cdot)$. Then we set

$$w_i := \frac{1}{(\widehat{S}_{\text{censor}}(Y_i))^2} \quad \text{for } i \in [n]. \tag{35}$$

The intuition is that the set $\mathcal{E}(t)$ is a biased sample of possible pairs: a point $(i, j) \in \mathcal{E}(t)$ must have point i not be censored. All else equal, had point i been censored instead (but had the same true survival time), then its survival time would still be less than that of point j, but this single change (whether point i was censored or not) would alter the value of C_t^{td}. In particular, $\mathcal{E}(t)$ "over-emphasizes" pairs (i, j) where point i is not censored, so we "up-weight" point i if its observed time Y_i is more likely to have been censored. Uno et al. [2011, Appendix A] make this argument precise by formally showing that C_t^{td} converges (as the amount of data goes to infinity) to

$$C_t^{\text{td}\,*} := \mathbb{P}\big(\widehat{S}(Y|X) < \widehat{S}(Y|X') \mid Y < t, Y < Y'\big).$$

Notice that this true population-level target does *not* depend on the event indicator Δ. This sort of weighting could be applied to the c-index and C^{td} scores too if one wishes to modify them so that they estimate population-level quantities that do not depend on Δ.

Remark 2.2 (Using validation or test data with C_t^{td})**.** As an important implementation detail, we had mentioned that the evaluation metrics we present can be computed using validation or test data. In the case of the C_t^{td} metric, we would learn both the time-to-event prediction model (that can predict $\widehat{S}(\cdot|x)$ for any $x \in \mathcal{X}$) and the censoring distribution's right tail probability function $\widehat{S}_{\text{censor}}(\cdot)$ from training data. After learning these functions, we treat them as fixed and evaluate C_t^{td} on validation or test data.

We caution that if $\widehat{S}_{\text{censor}}(\cdot)$ is a poor estimate of the true population-level $S_{\text{censor}}(t) = \mathbb{P}(C > t)$ function, then the C_t^{td} score with weights given by equation (35) may be suspect. Note that we have made the simplifying assumption that censoring time C is independent of raw input X. If this assumption does not hold, then we should replace $\widehat{S}_{\text{censor}}(\cdot)$ in equation (35) with a version that conditions on a given raw input (meaning that we instead aim to estimate the true population level function $S_{\text{censor}}(t|x) := \mathbb{P}(C > t|X = x)$). However, estimating $S_{\text{censor}}(\cdot|x)$ could be as difficult as estimating the conditional survival function $S(\cdot|x)$, so that it is possible that the weights w_i used with the C_t^{td} score in this case are unreliable.

Integrated truncated time-dependent concordance index. We point out that we could integrate C_t^{td} over time t to arrive at a single number. We would have to specify the limits of integration.

Definition 2.4 (Integrated truncated time-dependent concordance index)**.** Suppose that we have a survival function estimate $\widehat{S}(\cdot|x)$ for any $x \in \mathcal{X}$. Let $t_{\min} \geq 0$ and $t_{\max} > t_{\min}$ be user-specified lower and upper limits of integration. Then we define the integrated truncated time-dependent concordance index as

$$C^{\text{td}}_{[t_{\min}, t_{\max}]} := \frac{1}{t_{\max} - t_{\min}} \int_{t_{\min}}^{t_{\max}} C_u^{\text{td}} \mathrm{d}u.$$

This score is also between 0 and 1, where higher is better.

In practice, we numerically evaluate this integral along a time grid.

Time-dependent AUC. There are also time-dependent AUC scores [Uno et al., 2007, Hung and Chiang, 2010], which we will only briefly cover since they are very similar to the truncated time-dependent concordance index C_t^{td}. The key idea is that for a fixed time t, we set up a binary classification problem of distinguishing between points who died no later than time t (the "positive" class), and points who died after time t (the "negative" class). We take $\widehat{S}(t|\cdot) : \mathcal{X} \to [0,1]$ to be the probabilistic binary classifier that we aim to compute an AUC for. When this classifier predicts a lower survival probability for $x \in \mathcal{X}$, then this means that x is predicted to have a higher chance of being in the positive class.

Then to estimate the AUC, Hung and Chiang [2010] showed that we can use the same equation as the C_t^{td} index (equation (71)) except with two small changes. First, we replace $\mathcal{E}(t)$ with

$$\widetilde{\mathcal{E}}(t) := \{(i,j) \in [n] \times [n] : \Delta_i = 1, Y_i \le t, Y_j > t\}.$$

Notice that if (i,j) is in $\widetilde{\mathcal{E}}(t)$, then it means that point i is an example point from the "positive" class, and point j is an example point from the "negative" class. These are points we evaluate the binary classifier on. The second change is that we set the weight $w_i := 1/[\widehat{S}_{\text{censor}}(Y_i)\widehat{S}_{\text{censor}}(t)]$ (where we are simplifying Hung and Chiang's equation (3) for the case where the censoring distribution does not depend on the raw input).

Again, this approach uses weights w_i that depend on $\widehat{S}_{\text{censor}}(t)$ estimating $S_{\text{censor}}(t) = \mathbb{P}(C > t)$ accurately. If $\widehat{S}_{\text{censor}}(\cdot)$ is a poor estimate of $S_{\text{censor}}(\cdot)$, then the time-dependent AUC score could be unreliable.

There are many other ways to define a time-dependent AUC evaluation metric though. See the surveys by Blanche et al. [2013] and Lambert and Chevret [2016] for details.

2.5.2 Squared Error of the Predicted Survival Function

Brier score. There are accuracy metrics that more directly assess error of an estimated survival function $\widehat{S}(\cdot|x)$ without ranking. For example, the Brier score [Graf et al., 1999] is defined for a specific time $t \ge 0$ and aims to measure the error

$$\text{BS}^*(t) := \mathbb{E}[(\mathbb{1}\{T > t\} - \widehat{S}(t|X))^2],$$

where the expectation is over sampling X and T using the generative procedure in Section 2.1.

To empirically estimate $\text{BS}^*(t)$, we first consider if censoring never happens, so every point $i \in [n]$ has observed time Y_i equal to the true survival time T_i. Then we could estimate $\text{BS}^*(t)$ with

$$\text{BS}_{\text{no-censoring}}(t) := \frac{1}{n}\sum_{i=1}^{n}[(\mathbb{1}\{Y_i > t\} - \widehat{S}(t|X_i))^2].$$

To account for censoring, Graf et al. [1999] proposed the following approach. Similar to how we had defined the C_t^{td} index, we assume that censoring time C is independent of raw input X and estimate $S_{\text{censor}}(\cdot)$ with the Kaplan-Meier estimator to obtain $\widehat{S}_{\text{censor}}(\cdot)$. Then we define the Brier score error metric we use is as follows.

Definition 2.5 (Brier score)**.** Suppose that we have a survival function estimate $\widehat{S}(\cdot|x)$ for any $x \in \mathcal{X}$ and also an estimate $\widehat{S}_{\text{censor}}(\cdot)$ of $S_{\text{censor}}(\cdot)$. We define the Brier score at time $t \ge 0$ by

$$\text{BS}(t) := \frac{1}{n}\sum_{i=1}^{n}\left[\frac{\widehat{S}(t|X_i)^2\Delta_i\mathbb{1}\{Y_i \le t\}}{\widehat{S}_{\text{censor}}(Y_i)} + \frac{(1-\widehat{S}(t|X_i))^2\mathbb{1}\{Y_i > t\}}{\widehat{S}_{\text{censor}}(t)}\right],$$

which is nonnegative. Lower scores are better.

Graf [1998] showed that $\text{BS}(t)$ converges to $\text{BS}^*(t)$ as $n \to \infty$ with the additional assumption that $S_{\text{censor}}(t) > 0$. Separately, as a reminder, if we are computing the Brier score on validation or test data, then $\widehat{S}_{\text{censor}}(\cdot)$ is estimated using *training* data (as mentioned in Remark 2.2).

Just like with the truncated time-dependent concordance index and time-dependent AUC scores, Brier scores crucially depend on $\widehat{S}_{\text{censor}}(\cdot)$ estimating $S_{\text{censor}}(\cdot)$ well. As a reminder, we have assumed that censoring time C is independent of raw input X. If this assumption does not hold, then Brier scores (as we have defined them in Definition 2.5) are not guaranteed to be proper [Rindt et al., 2022].

Integrated Brier score. We could of course integrate the Brier score across time to arrive at a single number.

Definition 2.6 (Integrated Brier score)**.** Suppose that we have a survival function estimate $\widehat{S}(\cdot|x)$ for any $x \in \mathcal{X}$. Let $t_{\text{min}} \geq 0$ and $t_{\text{max}} > t_{\text{min}}$ be user-specified lower and upper limits of integration. The integrated Brier score is defined as

$$\text{IBS} := \frac{1}{t_{\text{max}} - t_{\text{min}}} \int_{t_{\text{min}}}^{t_{\text{max}}} \text{BS}(u) \text{d}u.$$

This score is also nonnegative, where lower scores are better.

2.5.3 Distribution Calibration

To assess how well-calibrated a predicted survival function $\widehat{S}(\cdot|x)$ is, Haider et al. [2020] proposed a calibration metric called Distribution Calibration (abbreviated "D-Calibration"). To explain how this works, Haider *et al.* consider $\widehat{S}(\cdot|x)$ to be perfectly calibrated if

$$\mathbb{P}\Big(\widehat{S}(T|X) \in [a, b]\Big) = b - a \quad \text{for all intervals } [a, b] \subseteq [0, 1], \tag{36}$$

where the probability is over randomness in sampling raw input X and its corresponding survival time T as stated in the generative procedure in Section 2.1.

Note that if we plug in the true survival function $S(\cdot|x)$ in place of the estimate $\widehat{S}(\cdot|x)$ into equation (36), then $S(\cdot|x)$ would be perfectly calibrated. This is a consequence of the *probability integral transform* result that states that for any continuous real-valued random variable A with CDF $G : \mathbb{R} \to [0, 1]$, the random variable $G(A)$ is uniformly distributed over $[0, 1]$.[12]

To turn condition (36) into a calibration metric that we can compute, we empirically evaluate it for some user-specified intervals. We first do this when there is no censoring. Note that the D-Calibration procedure that we are about to describe is a bit more involved than the previous evaluation metrics. As we shall see, it ultimately leads to thresholding on a p-value of a statistical test to decide whether the distribution corresponding to $\widehat{S}(\cdot|x)$ should be deemed calibrated or not.

The case without censoring. We define the following subset of data points for any interval $\mathcal{I} \subseteq [0, 1]$:

$$\mathcal{D}(\mathcal{I}) := \{i \in [n] : \Delta_i = 1, \widehat{S}(Y_i|X_i) \in \mathcal{I}\}.$$

In particular, point i being in $\mathcal{D}(a, b)$ means that point i died (so that $Y_i = T_i$) and the predicted survival probability $\widehat{S}(Y_i|X_i)$ is in the interval $\mathcal{I}$. Then we would like

$$\frac{|\mathcal{D}([a, b])|}{n} \approx b - a \quad \text{for } [a, b] \subseteq [0, 1].$$

[12] We are applying the probability integral transform result to the continuous random variable corresponding to T conditioned on $X = x$, which has CDF $F(\cdot|x)$. Thus, if we evaluate the CDF at random survival time T sampled from $\mathbb{P}_{T|X=x}$, *i.e.*, we compute the random variable $F(T|x)$, then this random variable is uniform over $[0, 1]$. Since $S(\cdot|x) = 1 - F(\cdot|x)$, we conclude that $S(T|x)$ is also uniformly distributed over $[0, 1]$. This analysis holds for any realization x that could be sampled from $\mathbb{P}_X$ so that accounting for randomness in sampling $X = x$, we still get that $S(T|X)$ is uniform over $[0, 1]$.

We evaluate this condition for some pre-specified equal-width intervals that cover the whole interval $[0, 1]$. For example, we could use the 10 bins $\mathcal{I}_1 := [0, 0.1], \mathcal{I}_2 := (0.1, 0.2], \ldots, \mathcal{I}_{10} := (0.9, 1]$ (the intervals do not have to be closed on both ends). Here, we would like

$$\frac{|\mathcal{D}(\mathcal{I}_\ell)|}{n} \approx \frac{1}{10} \quad \text{for all } \ell \in [10],$$

meaning that we want to check that the 10 probabilities $\widehat{p}_1 := \frac{|\mathcal{D}(\mathcal{I}_1)|}{n}, \ldots, \widehat{p}_{10} := \frac{|\mathcal{D}(\mathcal{I}_{10})|}{n}$ are close to a uniform distribution. Haider *et al.* suggest using a standard chi-squared test to check for uniformity, which in this case corresponds to computing the test statistic

$$\chi^2 := 10n \sum_{\ell=1}^{10} \left(\widehat{p}_\ell - \frac{1}{10}\right)^2 = 10n \sum_{\ell=1}^{10} \left(\frac{|\mathcal{D}(\mathcal{I}_\ell)|}{n} - \frac{1}{10}\right)^2.$$

We then compute

$$\text{p-value} := \mathbb{P}(\text{chi-squared variable with 9 degrees of freedom} \geq \chi^2).$$

If the p-value is at least some user-specified threshold (Haider et al. [2020] use 0.05), then we declare the distribution to be uniform, so the predicted survival function $\widehat{S}$ is considered calibrated. Otherwise, we consider $\widehat{S}$ to not be calibrated. For a continuous version that is not binary, we could, for instance, use the χ^2 test statistic (Goldstein et al. [2020] use what we have written for the χ^2 test statistic but they exclude the $10n$ scale factor).

Accounting for censoring. By how we have defined the set $\mathcal{D}(\mathcal{I})$ above, note that probabilities $\widehat{p}_1, \ldots, \widehat{p}_{10}$ that we checked against a uniform distribution would not sum to 1 if there are censored data:

$$\sum_{\ell=1}^{m} \widehat{p}_{(\ell)} = \sum_{\ell=1}^{m} \frac{|\mathcal{D}(\mathcal{I}_\ell)|}{n} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{\Delta_i = 1\} = 1 - \underbrace{\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{\Delta_i = 0\}}_{\substack{\text{fraction of data} \\ \text{that are censored}}}.$$

The idea then is that we will modify the estimated probabilities $\widehat{p}_1, \ldots, \widehat{p}_{10}$, increasing them in a particular way so that they form a valid probability distribution and that they use information from the censored data.

Prior to doing any modification, remember that for each interval $\ell \in [10]$,

$$\widehat{p}_\ell = \frac{|\mathcal{D}(\mathcal{I}_\ell)|}{n} = \sum_{i \in \mathcal{D}(\mathcal{I}_\ell)} \frac{1}{n}.$$

The right-most expression suggests the following interpretation: each training point in $\mathcal{D}(\mathcal{I}_\ell)$ (i.e., each training point that is assigned to interval ℓ) contributes a probability mass of $1/n$ to interval ℓ.

We now view each censored point to also have probability mass $1/n$, but we want to figure out how to allocate this probability mass to the 10 different intervals. Haider et al. [2020] proposed the following strategy. For each point $i \in [n]$ that is censored (*i.e.*, $\Delta_i = 0$), we look at which interval $\widehat{S}(Y_i|X_i)$ falls into among $\mathcal{I}_1, \ldots, \mathcal{I}_{10}$. Denote the resulting interval's index as $\widetilde{\ell} \in [10]$. We then distribute the probability mass $1/n$ evenly among intervals $\widetilde{\ell}, \widetilde{\ell} + 1, \ldots, 10$. In other words, for censored point i, we update

$$\widehat{p}_\ell \leftarrow \widehat{p}_\ell + \frac{1}{n} \cdot \left(\frac{1}{10 - \widetilde{\ell} + 1}\right) \quad \text{for } \ell \in \{\widetilde{\ell}, \widetilde{\ell} + 1, \ldots, 10\}.$$

After iterating through all censored points and distributing each of their $1/n$ probability mass to the intervals in the above manner, the resulting probabilities $\widehat{p}_1, \ldots, \widehat{p}_{10}$ will indeed sum to 1, and we proceed with the chi-squared test as before to determine if the distribution is calibrated.

2.5.4 Error Metrics for Survival Time Point Estimates

Aside from the original c-index evaluation metric, all the other evaluation metrics we covered so far evaluated predicted survival functions $\widehat{S}(\cdot|\cdot)$ for different raw inputs at different times. Now we turn to evaluation metrics specific to when we predict a single survival time number per data point.

For point i with predicted survival function $\widehat{S}(\cdot|X_i)$, we denote its predicted survival time as $\widehat{T}_i$, which we could take to be the median survival time estimate (the time at which $\widehat{S}(\cdot|X_i)$ crosses probability $1/2$) or the mean survival time estimate (area under the function $\widehat{S}(\cdot|X_i)$).

If point i is not censored, then we could easily use any standard regression error metric to compare Y_i and $\widehat{T}_i$, such as squared error $(Y_i - \widehat{T}_i)^2$ or absolute error $|Y_i - \widehat{T}_i|$. If point i is censored, then we have no ground truth survival time to compare against. A naive solution is to use the so-called *hinge* error, for which the squared error version is $(Y_i - \widehat{T}_i)^2 \mathbb{1}\{Y_i > \widehat{T}_i\}$ and the absolute error version is $(Y_i - \widehat{T}_i)\mathbb{1}\{Y_i > \widehat{T}_i\}$; these only give nonzero error when the predicted survival time $\widehat{T}_i$ is less than the observed time Y_i (which we know to be when the point is still alive since it is a censoring time).

We state two strategies for dealing with censored data that have been shown to often work well. Both involve estimating a pseudo "ground truth" survival time to compare the predicted survival time against for each censored data point (put another way, we are imputing the ground truth survival times for censored data). Let $\widehat{S}_{\text{KM}}(\cdot)$ denote the Kaplan-Meier estimate of the population-level survival function $S_{\text{pop}}(t) := \mathbb{P}(T > t)$ (as given in equation (33)). Note that this Kaplan-Meier estimator is fitted to the *training* data, meaning that even if we are evaluating error on validation or test data that are not the same as the training data, in the equations that follow, $\widehat{S}_{\text{KM}}(\cdot)$ is fitted to the training data (this is similar to what we stated in Remark 2.2 with the difference being that we are now estimating $S_{\text{pop}}(t) = \mathbb{P}(T > t)$ and not $S_{\text{censor}}(t) = \mathbb{P}(C > t)$).

The *margin* [Haider et al., 2020] approach estimates the pseudo ground truth survival time of a censored point $i \in [n]$ to be

$$T_i^{\text{margin}} := Y_i + \frac{\int_{Y_i}^{\infty} \widehat{S}_{\text{KM}}(u)\mathrm{d}u}{\widehat{S}_{\text{KM}}(Y_i)},$$

where we numerically evaluate the integral. The right-hand side aims to estimate $\mathbb{E}[T_i|T_i > Y_i]$.[13]

Meanwhile, the *pseudo-observation* (PO) approach [Qi et al., 2023] instead estimates the ground truth survival time of a censored point $i \in [n]$ to be

$$T_i^{\text{PO}} := (n+1) \underbrace{\int_0^{\infty} \widehat{S}_{\text{KM}^{+i}}(u)\mathrm{d}u}_{\substack{\text{estimate of mean survival} \\ \text{time including evaluation} \\ \text{point } i}} - n \underbrace{\int_0^{\infty} \widehat{S}_{\text{KM}}(u)\mathrm{d}u}_{\substack{\text{estimate of mean survival} \\ \text{time excluding evaluation} \\ \text{point } i}}, \tag{37}$$

where $\widehat{S}_{\text{KM}^{+i}}(\cdot)$ refers to the Kaplan-Meier estimator fitted to the training data with evaluation point i included as an additional training point. Once again, the integrals are numerically evaluated. The first integral is an estimate of the mean survival time of survival function $\widehat{S}_{\text{KM}^{+i}}(\cdot)$, and the second integral is an estimate of the mean survival time of survival function $\widehat{S}_{\text{KM}}(\cdot)$. Taking this difference is based on the bias-corrected jackknife estimator.[14]

[13] Recall from Section 2.2.1 that integrating a survival function from time 0 to time ∞ yields the mean survival time. Integrating a survival function instead from time Y_i to time ∞ and dividing by $\mathbb{P}(T_i > Y_i)$ yields the mean survival time conditioned on survival beyond time Y_i.

[14] As a technical remark, our exposition here of the PO approach follows how Qi et al. [2024a] have currently implemented it in their GitHub repository rather than how they have originally stated it in their paper and in their earlier work [Qi et al., 2023]. The difference is that originally, Qi et al. [2023] define $T_i^{\text{PO}} := n\int_0^{\infty} \widehat{S}_{\text{KM}}(u)\mathrm{d}u - (n-1)\int_0^{\infty} \widehat{S}_{\text{KM}^{-i}}(u)\mathrm{d}u$, where $\widehat{S}_{\text{KM}}$ is assumed to be fitted to a dataset that includes evaluation point i, and $\widehat{S}_{\text{KM}^{-i}}$ is the version of the Kaplan-Meier estimator fitted to the dataset excluding evaluation point i.

After computing one of these pseudo ground truth labels, we could treat a censored point's pseudo ground truth survival time as if it were a real ground truth survival time and evaluate standard regression error metrics like squared error or absolute error. For example, the mean absolute error metric using the PO approach would be given by

$$\text{MAE-PO} := \frac{1}{n}\sum_{i=1}^{n}\big(\Delta_i \underbrace{|\widehat{T}_i - Y_i|}_{\substack{\text{when uncensored,}\\ \text{use observed time}}} + (1-\Delta_i) \underbrace{|\widehat{T}_i - T_i^{\text{PO}}|}_{\substack{\text{when censored, use}\\ \text{pseudo ground truth}}}\big), \tag{38}$$

where, as a reminder, $\widehat{T}_i$ is the predicted survival time of the i-th point using the time-to-event prediction model that we are evaluating (possibly where we convert a predicted survival function into a point estimate by backing out a median or mean survival time estimate).

Haider et al. [2020] explained that taking an equally weighted average as in equation (38) may not be a good idea, as we may be more confident in the (pseudo) ground truth values for some points vs others. The intuition is as follows. Suppose that we are measuring survival times of people in years, and that for the population under consideration, no one has a survival time greater than 130 years. Imagine that a data point (corresponding to a person) was censored at time 0 (and censoring times are independent of survival times). Then we know very little about what the true survival time should be, and the pseudo ground truth value computed (whether using the margin, PO, or some other approach altogether) would likely be unreliable. In contrast, suppose that a data point was censored at 110 years. For this data point, we would be much more confident about the pseudo ground truth value being close to the true value. With this intuition, Haider *et al.* suggested that for data points that are censored, we should give higher weights to points that are censored later. They operationalize this intuition by using a weighted mean absolute error metric

$$\begin{aligned}&\text{weighted-MAE-PO}\\ &:= \frac{1}{\sum_{i=1}^{n} w_i}\sum_{i=1}^{n} w_i\big(\Delta_i \underbrace{|\widehat{T}_i - Y_i|}_{\substack{\text{when uncensored,}\\ \text{use observed time}}} + (1-\Delta_i) \underbrace{|\widehat{T}_i - T_i^{\text{PO}}|}_{\substack{\text{when censored, use}\\ \text{pseudo ground truth}}}\big),\end{aligned}$$

where they assign weights as follows:

$$w_i := \begin{cases} 1 & \text{if } \Delta_i = 1,\\ 1 - \widehat{S}_{\text{KM}}(Y_i) & \text{if } \Delta_i = 0.\end{cases}$$

For a thorough experimental evaluation of these pseudo ground truth evaluation metrics, and also for details on why using (weighted) mean absolute error makes sense in many time-to-event prediction tasks, see the paper by Qi et al. [2023].

Meanwhile, Qi et al. [2023] also showed that MAE with median survival times is a proper scoring rule for *uncensored* datasets. However, this theoretical result is unsatisfying in that the main technical hurdle in survival analysis is censoring.

2.6 Additional Remarks on Classification and Regression

Although we already discussed how the time-to-event prediction in discrete time relates to binary classification at different time indices (Section 2.3), the models we had derived using maximum likelihood (DeepHit, Nnet-survival, Kaplan-Meier and Nelson Aalen estimators) were not just existing off-the-shelf binary classifiers. In this section, we discuss an approach called *survival stacking* [Craig et al., 2021] that converts any time-to-event prediction problem with raw input space $\mathcal{X} = \mathbb{R}^d$ into a binary classification problem such that we can use any off-the-shelf probabilistic binary classifier for prediction such as logistic regression or random forests (Section 2.7). This conversion fundamentally models time to be discrete and can be quite expensive: with n training points that each have d features, the input training feature matrix (each row is a feature vector)

could be viewed as a 2D table that is n-by-d. After converting the problem using survival stacking, the training feature matrix for the binary classifier could have as many as $\mathcal{O}(n^2)$ rows and $\mathcal{O}(d+n)$ columns.

Separately, we relate time-to-event prediction to the classical regression setup (Section 2.7.1). This relationship is more straightforward and considers what happens if censoring did not happen.

2.7 Survival Stacking: Converting Time-to-Event Prediction to Binary Classification

To explain survival stacking, we follow Craig et al. [2021] and provide an example of how survival stacking converts a small toy time-to-event prediction problem with $n = 3$ and $\mathcal{X} = \mathbb{R}^d$ with $d = 2$ into a binary classification problem (we use their same toy example, although we use the notation that we have introduced in this monograph).

Specifically, suppose that we have three training points $(X_1, Y_1, \Delta_1), (X_2, Y_2, \Delta_2), (X_3, Y_3, \Delta_3)$, where for ease of exposition, we assume that these points are sorted so that $Y_1 < Y_2 < Y_3$. Suppose that $\Delta_1 = 1$, $\Delta_2 = 0$, and $\Delta_3 = 1$. In terms of notation, each X_i is in $\mathbb{R}^2$, for which we write $X_i = (X_{i,1}, X_{i,2}) \in \mathbb{R}^2$. Thus, we could view our training data as follows:

$$
\begin{array}{c}
\\
\text{point 1} \\
\text{point 2} \\
\text{point 3}
\end{array}
\begin{array}{c}
\begin{array}{cc} \text{feature 1} & \text{feature 2} \end{array} \\
\begin{bmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \\ X_{3,1} & X_{3,2} \end{bmatrix}
\end{array}
\qquad
\begin{array}{c}
\begin{array}{cc} \text{observed time} & \text{event indicator} \end{array} \\
\begin{bmatrix} Y_1 & 1 \\ Y_2 & 0 \\ Y_3 & 1 \end{bmatrix}
\end{array}
$$

Enumerate the unique times of death. To begin the survival stacking conversion process, we enumerate the unique times in which death occurred, which in this case is $\tau_{(1)} = Y_1$ and $\tau_{(2)} = Y_3$ (we are intentionally reusing our notation from earlier for modeling discrete time, where the time grid points are denoted $\tau_{(1)} < \cdots < \tau_{(L)}$). In this case, there are $L = 2$ unique times of death.

Collect information from time index 1. At time index 1 (corresponding to time $\tau_{(1)} = Y_1$), we list all the data points that are "at risk" (could still possibly die) at that time. In this case, since $Y_1 < Y_2 < Y_3$, at time $\tau_{(1)} = Y_1$, all three data points are at risk. What we do then is we create the following 2D table:

$$
\mathbf{X}_{(1)} := \begin{array}{c}
\\
\text{point 1} \\
\text{point 2} \\
\text{point 3}
\end{array}
\begin{array}{c}
\begin{array}{cc|cc} \text{feature 1} & \text{feature 2} & \text{time index 1} & \text{time index 2} \end{array} \\
\left[\begin{array}{cc|cc} X_{1,1} & X_{1,2} & 1 & 0 \\ X_{2,1} & X_{2,2} & 1 & 0 \\ X_{3,1} & X_{3,2} & 1 & 0 \end{array}\right]
\end{array}
$$

In particular, the number of rows of $\mathbf{X}_{(1)}$ is the number of points at risk at time index 1 (corresponding to time $\tau_{(1)}$) while the number of columns is $d + L = 2 + 2 = 4$. We have added new columns that simply indicate which time index we are currently looking at. Next, we also create the following vector that indicates which of the points at risk actually died at time index 1, which would only be training point 1:

$$
\mathbf{y}_{(1)} := \begin{array}{c}
\\
\text{point 1} \\
\text{point 2} \\
\text{point 3}
\end{array}
\begin{array}{c}
\text{death at time index 1} \\
\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}
\end{array}
$$

Collect information from time index 2. Now we proceed to time index 2 (corresponding to time $\tau_{(2)} = Y_3$) and repeat the same idea. We list all the data points at risk, which would just be data

point 3. We create the following 2D table:

$$\mathbf{X}_{(2)} := \begin{array}{c} \\ \text{point 3} \end{array} \begin{array}{c} \begin{array}{cc|cc} \text{feature 1} & \text{feature 2} & \text{time index 1} & \text{time index 2} \end{array} \\ \begin{bmatrix} X_{3,1} & X_{3,2} & 0 & 1 \end{bmatrix} \end{array}$$

We also create a vector indicating that point 3 did experience death at time index 2:

$$\mathbf{y}_{(2)} := \begin{array}{c} \\ \text{point 3} \end{array} \begin{array}{c} \begin{array}{c} \text{death at time index 2} \end{array} \\ \begin{bmatrix} 1 \end{bmatrix} \end{array}$$

Stack information vertically to get training data for classifier. At this point, we have gone through all the unique times of death. We vertically stack $\mathbf{X}_{(1)}$ and $\mathbf{X}_{(2)}$, and similarly we vertically stack $\mathbf{y}_{(1)}$ and $\mathbf{y}_{(1)}$ to get:

$$\mathbf{X} := \left[\begin{array}{c} \mathbf{X}_{(1)} \\ \hline \mathbf{X}_{(2)} \end{array}\right] = \begin{bmatrix} X_{1,1} & X_{1,2} & 1 & 0 \\ X_{2,1} & X_{2,2} & 1 & 0 \\ X_{3,1} & X_{3,2} & 1 & 0 \\ X_{3,1} & X_{3,2} & 0 & 1 \end{bmatrix}$$

$$\mathbf{y} := \left[\begin{array}{c} \mathbf{y}_{(1)} \\ \hline \mathbf{y}_{(2)} \end{array}\right] = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Then we treat the rows of $\mathbf{X}$ as feature vectors, where the i-th row has binary classification label given by the i-th entry of $\mathbf{y}$, and we use these to train a probabilistic binary classifier of our choosing, such as logistic regression or a random forest.

The binary classification task. To help make sense of what this binary classifier is predicting, consider the fourth training feature vector being fed in, *i.e.*, the last row of $\mathbf{X}$: $(X_{3,1}, X_{3,2}, 0, 1)$. The classifier is being told what the original feature vector is for this data point (namely $X_3 = (X_{3,1}, X_{3,2})$) along with which time index we are making a prediction for (time index 2, encoded as the vector $(0, 1)$), which of course implies that the data point is still alive at this particular time step. We are thus predicting the probability of death, assuming that the point is still at risk. In other words, we are predicting the discrete time hazard probability at time index 2.

More generally, what is happening is that we are discretizing time using the unique times of death $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$. Let $e_\ell \in \{0,1\}^L$ denote an L-dimensional vector that is all zeros except for a single 1 at the ℓ-th entry, where $\ell \in [L]$. For any test feature vector $x \in \mathbb{R}^d$ in the original space, let $x \oplus e_\ell$ denote the concatenation of vectors x and e_ℓ (so $x \oplus e_\ell$ is in $\mathbb{R}^{d+L}$). Then the binary classifier, given input feature vector $x \oplus e_\ell$, predicts the hazard probability corresponding to the original feature vector x at time index $\ell \in [L]$, *i.e.*, what we had denoted as $h[\ell|x]$ in our earlier coverage. When the binary classifier is logistic regression, Craig et al. [2021, Section 2.3] show that the resulting classifier closely approximates the original Cox proportional hazards model [Cox, 1972]. We discuss the Cox model later in Section 3.3.

Training dataset size for the binary classifier. In the toy example above, we ended up with a classification training feature matrix $\mathbf{X}$ that is 4-by-4 even though the original time-to-event prediction feature matrix was only 3-by-2. How much larger could the classification training feature matrix be?

We can compute the exact size of $\mathbf{X}$. In general, the number of columns of $\mathbf{X}$ is $d + L$ where, as a reminder, L is the number of unique times of death. As for the number of rows of $\mathbf{X}$, we can determine this by adding up the number of rows of $\mathbf{X}_{(1)}$, $\mathbf{X}_{(2)}$, and so forth (which we had vertically stacked to form $\mathbf{X}$). The number of rows in each $\mathbf{X}_{(\ell)}$ (for $\ell \in [L]$) is precisely the number of points

at risk at time index ℓ, given by $N[\ell] = \mathbb{1}\{Y_j \geq \tau_{(\ell)}\}$ from equation (30). Thus, in general,

$$\text{number of rows in } \mathbf{X} = \sum_{\ell=1}^{L} N[\ell] = \sum_{\ell=1}^{L} \sum_{j=1}^{n} \mathbb{1}\{Y_j \geq \tau_{(\ell)}\}.$$

In the worst case, every observed time is unique and is a time of death (so that at each time index, exactly one point dies). In this case, one could see that at the first time index there are n points at risk (so $\mathbf{X}_{(1)}$ would have n rows), at the second time index there are $n-1$ points at risk (so $\mathbf{X}_{(2)}$ would have $n-1$ rows), *etc*. In this case, the number of rows of $\mathbf{X}$ is $n + (n-1) + (n-2) + \cdots + 1 = \frac{n(n+1)}{2}$. Meanwhile, if every observed time is unique, then it means that $L = n$, so the number of columns of $\mathbf{X}$ is $d + L = d + n$. Thus, the worst case size of $\mathbf{X}$ is $\frac{n(n+1)}{2} = \mathcal{O}(n^2)$ rows by $d + n$ columns, which of course is larger than the time-to-event prediction feature matrix size of n-by-d.

In practice, to prevent the stacked matrix $\mathbf{X} \in \mathbb{R}^{(\sum_{\ell=1}^{L} N[\ell]) \times (d+L)}$ from being too large, what one could do is discretize the time grid to be coarser: instead of taking L to be the number of unique times of death, we could manually specify L to be much smaller than n and discretize Y_i values to L time indices (we presented some ways of doing this in Section 2.3.2). In doing so, we would directly control the number of columns of $\mathbf{X}$ (since we would set the value of L and the number of columns is $d + L$), while also decreasing the number of rows of $\mathbf{X}$. Note that dealing with the number of rows of $\mathbf{X}$ being large is somewhat less of an issue in that many classifiers are designed to scale to large datasets (*e.g.*, XGBoost [Chen and Guestrin, 2016]). Classifiers that support minibatch training could avoid looking at all rows of $\mathbf{X}$ at once.

2.7.1 Connection to Regression: Conditional CDF Estimation

The connection to regression is straightforward: consider the case where censoring happens with probability 0 (which could be thought of as an extreme case where the censoring time C is deterministically $+\infty$ in the generative procedure from Section 2.1). Then our training data would be the same as that of standard regression (we could ignore the event indicator Δ_i variables since they would all be equal to 1) although all the regression labels (the Y_i variables) are guaranteed to be nonnegative (whereas in, for instance, linear regression, the regression labels are not constrained to be nonnegative). The prediction task of estimating the conditional survival function without censoring would simply amount to conditional CDF estimation for a regression label, which has previously been studied (*e.g.*, Chagny and Roche 2014).

2.A Technical Details

2.A.1 Definition of the Raw Input Space

As stated in Section 2.1, we assume that the raw input space $\mathcal{X}$ is the "support" of distribution $\mathbb{P}_X$. Roughly, the support of $\mathbb{P}_X$ consists of all possible values that we could sample from $\mathbb{P}_X$. We now build up to formally defining what the support of a distribution is.

First, to motivate why $\mathcal{X}$ cannot be defined arbitrarily, consider the following toy example. Suppose that we set $\mathcal{X} := \mathbb{R}$, and that $\mathbb{P}_X$ is uniform over the unit interval $[0, 1]$. However, we aim to be able to make predictions (*i.e.*, to estimate one of the target functions that fully characterize $\mathbb{P}_{T|X}(\cdot|x)$) for all $x \in \mathcal{X}$. In the toy example given, $\mathbb{P}_{T|X}(\cdot|x)$ would not actually be defined when x is outside of $[0, 1]$ (*e.g.*, we cannot condition on $X = 2$). The fix is simple in this case: we should instead define $\mathcal{X} := [0, 1]$.

In general, we should set $\mathcal{X}$ to be the *support* of distribution $\mathbb{P}_X$, which we denote as $\text{supp}(\mathbb{P}_X)$. As concrete examples:

- If X is a discrete random vector in $\mathbb{R}^d$, then

$$\text{supp}(\mathbb{P}_X) := \{x \in \mathbb{R}^d : \mathbb{P}(X = x) > 0\}.$$

- If X is a continuous random vector over $\mathbb{R}^d$ with PDF $f_X(\cdot)$, then

$$\text{supp}(\mathbb{P}_X) := \overline{\{x \in \mathbb{R}^d : f_X(x) > 0\}},$$

  where the line over the set indicates that we are taking its closure.

However, even if X resides in $\mathbb{R}^d$ so that it is a fixed-length feature vector, in real applications, X could consist of a mix of discrete and continuous features. In this case, the above definitions that require all features to be discrete or all features to be continuous are not adequate. A general definition of the support of $\mathbb{P}_X$ that works whenever X takes on a value in $\mathbb{R}^d$ is as follows:

$$\text{supp}(\mathbb{P}_X) := \{x \in \mathbb{R}^d : \mathbb{P}(\|X - x\| \leq r) > 0 \text{ for all } r > 0\},$$

where $\|\cdot\|$ denotes Euclidean distance.

Since neural networks can accommodate input spaces $\mathcal{X}$ that are not $\mathbb{R}^d$, we now substantially generalize the definition of $\text{supp}(\mathbb{P}_X)$. Specifically, suppose that $\mathbb{P}_X$ is defined over a separable metric space $(\mathcal{X}, \rho)$, where the function $\rho : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ is a metric: for any two $x, x' \in \mathcal{X}$, $\rho(x, x')$ gives a distance between x and x' (if $\mathcal{X}$ is Euclidean space, then we could take ρ to be Euclidean distance). Then we define

$$\text{supp}(\mathbb{P}_X) := \{x \in \mathcal{X} : \mathbb{P}(\rho(X, x) \leq r) > 0 \text{ for all } r > 0\}.$$

The reason we assumed that the metric space is separable is to guarantee that $\mathbb{P}\big(X \in \text{supp}(\mathbb{P}_X)\big) = 1$ [Cover and Hart, 1967].

In summary, by defining $\mathcal{X} := \text{supp}(\mathbb{P}_X)$, we can indeed condition on $X = x$ for all $x \in \mathcal{X}$, and in particular, we can then work with the conditional distribution $\mathbb{P}_{T|X}(\cdot|x)$ for all $x \in \mathcal{X}$.

2.A.2 Proof of Proposition 2.1: $H[\ell|x]$ as a First-Order Taylor Approximation of $-\log S[\ell|x]$

Recall the Taylor series expansion $-\log(1-z) = \sum_{p=1}^{\infty} \frac{z^p}{p}$ for $z \in [0, 1)$. Then assuming that $h[\ell|x] \in [0, 1)$ for all $\ell \in [L]$, we have

$$\begin{aligned}
-\log S[\ell|x] &= -\log \prod_{m=1}^{\ell} (1 - h[m|x]) \\
&= \sum_{m=1}^{\ell} -\log(1 - h[m|x]) \\
&= \sum_{m=1}^{\ell} \sum_{p=1}^{\infty} \frac{(h[m|x])^p}{p} \\
&= \sum_{m=1}^{\ell} \left(h[m|x] + \sum_{p=2}^{\infty} \frac{(h[m|x])^p}{p} \right) \\
&= H[\ell|x] + \sum_{m=1}^{\ell} \sum_{p=2}^{\infty} \frac{(h[m|x])^p}{p}.
\end{aligned}$$

□

2.A.3 Hazard Function Maximum Likelihood Derivation for the Kaplan-Meier and Nelson-Aalen Estimators

In this section, we derive equation (28):

$$\widehat{\theta}_\ell := \arg\max_{\theta_\ell} \mathcal{L}_{(\ell)}(\theta_\ell) = \frac{D[\ell]}{N[\ell]} \quad \text{for } \ell \in [L], \qquad \text{(28, reproduced)}$$

where, as a reminder:

$$\mathcal{L}_{(\ell)}(\theta_\ell) := \sum_{i=1}^{n} \mathbb{1}\{\ell \le \kappa(Y_i)\}\big\{B_{i,\ell}\log\theta_\ell + (1-B_{i,\ell})\log(1-\theta_\ell)\big\}, \qquad \text{(26, partially reproduced)}$$

$$D[\ell] := \sum_{j=1}^{n} \mathbb{1}\{Y_j = \tau_{(\ell)}\}\Delta_j = \sum_{j=1}^{n} B_{j,\ell}, \qquad \text{(29, reproduced)}$$

$$N[\ell] := \sum_{j=1}^{n} \mathbb{1}\{Y_j \ge \tau_{(\ell)}\} = \sum_{j=1}^{n} \mathbb{1}\{\kappa(Y_j) \ge \ell\}. \qquad \text{(30, reproduced)}$$

We set the derivative of $\widehat{\theta}_\ell$ with respect to θ_ℓ to 0 (for the moment, we do not worry about the constraint that $\theta_\ell \in [0,1]$; as we shall see shortly, the value of θ_ℓ that achieves derivative 0 is guaranteed to be between 0 and 1). We have

$$\begin{aligned}
0 &= \Big[\frac{\mathrm{d}\log\mathcal{L}_{(\ell)}(\theta_\ell)}{\mathrm{d}\theta_\ell}\Big]_{\theta_\ell=\widehat{\theta}_\ell} \\
&= \sum_{i=1}^{n} \mathbb{1}\{\ell \le \kappa(Y_i)\}\Big[\frac{B_{i,\ell}}{\widehat{\theta}_\ell} - \frac{1-B_{i,\ell}}{1-\widehat{\theta}_\ell}\Big] \\
&= \frac{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}B_{i,\ell}}{\widehat{\theta}_\ell} - \frac{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}(1-B_{i,\ell})}{1-\widehat{\theta}_\ell} \\
&= \frac{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}B_{i,\ell}}{\widehat{\theta}_\ell} \\
&\quad - \frac{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\} - \sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}B_{i,\ell}}{1-\widehat{\theta}_\ell}.
\end{aligned}$$

Rearranging terms, we have

$$\begin{aligned}
&\frac{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}B_{i,\ell}}{\widehat{\theta}_\ell} = \frac{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\} - \sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}B_{i,\ell}}{1-\widehat{\theta}_\ell} \\
\iff \widehat{\theta}_\ell &= \frac{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}B_{i,\ell}}{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}} \\
&= \frac{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}\Delta_i\mathbb{1}\{\kappa(Y_i) = \ell\}}{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}} \\
&= \frac{\sum_{i=1}^{n}\Delta_i\mathbb{1}\{\kappa(Y_i) = \ell\}}{\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}} \\
&= \frac{D[\ell]}{N[\ell]}.
\end{aligned}$$

Note that $\widehat{\theta}_\ell = \frac{D[\ell]}{N[\ell]}$ is guaranteed to be between 0 and 1. To verify that $\widehat{\theta}_\ell$ is indeed the maximum and not the minimum, we check that $[\frac{\mathrm{d}\mathcal{L}_{(\ell)}(\theta_\ell)}{\mathrm{d}\theta_\ell^2}]_{\theta_\ell=\widehat{\theta}_\ell} < 0$. In fact,

$$\frac{\mathrm{d}\mathcal{L}_{(\ell)}(\theta_\ell)}{\mathrm{d}\theta_\ell^2} = -\sum_{i=1}^{n}\mathbb{1}\{\ell \le \kappa(Y_i)\}\Big[\frac{B_{i,\ell}}{\theta_\ell^2} + \frac{1-B_{i,\ell}}{(1-\theta_\ell)^2}\Big] < 0$$

for all $\theta_\ell \in (0,1)$. This finishes the proof. □

3 Deep Proportional Hazards Models

In this section, we cover perhaps the most widely used family of time-to-event prediction models used in practice, called proportional hazards models. Our exposition goes over a fairly general

formulation that includes, as special cases, the original Cox proportional hazards model [Cox, 1972] as well as its deep learning variant called DeepSurv [Faraggi and Simon, 1995, Katzman et al., 2018].

When determining what time-to-event prediction model to use in real applications, Cox models (*e.g.*, the original version or DeepSurv) are good baselines to try, similar to how logistic regression is good to try for binary classification and linear regression is good to try for predicting a continuous outcome. However, much like how logistic regression and linear regression make strong assumptions, Cox models do as well, which is why they often do not achieve state-of-the-art prediction accuracy.

In general, proportional hazards models assume that the hazard function factorizes as

$$h(t|x) = \mathbf{h}_0(t;\theta)e^{\mathbf{f}(x;\theta)} \quad \text{for } t \geq 0, x \in \mathcal{X}, \tag{39}$$

where we have two functions to be learned: the so-called *baseline hazard function* $\mathbf{h}_0(\cdot;\theta) : [0,\infty) \to [0,\infty)$ and the *log partial hazard function* $\mathbf{f}(\cdot;\theta) : \mathcal{X} \to \mathbb{R}$, both of which have parameter variable θ. The factorization implies that regardless of what the input x is, the hazard function $h(\cdot|x)$ must be proportional to the baseline hazard $\mathbf{h}_0(\cdot;\theta)$, which is why equation (39) is called the *proportional hazards assumption*. As we shall see in Section 3.1, the proportional hazards assumption imposes a strict constraint on the shapes of survival functions $S(\cdot|x)$. *The proportional hazard assumption often does not hold in real data, which is why proportional hazards models often do not work as well as more flexible time-to-event prediction models.*

The log partial hazard function $\mathbf{f}(\cdot;\theta)$ maps each input $x \in \mathcal{X}$ to a single real number that could be thought of as a risk score. A higher value of $\mathbf{f}(x;\theta)$ implies that the survival time tends to be lower. Since all inputs share the same dependence on time that is captured by the baseline hazard $\mathbf{h}_0(\cdot;\theta)$, under the proportional hazard assumption, the only difference between inputs $x, x' \in \mathcal{X}$ is captured entirely in comparing their log partial hazard function values $\mathbf{f}(x;\theta)$ and $\mathbf{f}(x';\theta)$. Thus, proportional hazards models could fundamentally be viewed as providing a way to rank data points based on the "scoring" function $\mathbf{f}(\cdot;\theta)$. We refer to $\mathbf{f}(\cdot;\theta)$ as the log partial hazard function because

$$\log h(t|x) = \log \mathbf{h}_0(t;\theta) + \mathbf{f}(x;\theta),$$

so $\mathbf{f}(x;\theta)$ only captures part of the full log hazard.

As a reminder, we saw two examples of proportional hazards models in Section 2, where we had assumed that $\mathcal{X} = \mathbb{R}^d$:

- In the exponential time-to-event prediction model (Examples 2.1 and 2.2), we had $\theta = (\beta, \psi) \in \mathbb{R}^d \times \mathbb{R}$, $\mathbf{h}_0(t;\theta) = e^{\psi}$, and $\mathbf{f}(x;\theta) = \beta^\top x$.
- In the Weibull time-to-event prediction model (Example 2.3), we instead had $\theta = (\beta, \psi, \phi) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$, $\mathbf{h}_0(t;\theta) = t^{e^{\phi}-1}e^{\psi+\phi}$, and $\mathbf{f}(x;\theta) = e^{\phi}\beta^\top x$.

These models are considered parametric proportional hazards models because the baseline hazard $\mathbf{h}_0(\cdot;\theta)$ and the log partial hazard $\mathbf{f}(\cdot;\theta)$ are assumed to have parametric forms. We had already shown in Section 2 how these two models could be learned via maximum likelihood (*i.e.*, how to obtain an estimate $\widehat{\theta}$ of θ) and how to subsequently predict hazard, cumulative hazard, and survival functions for these models. Basically, after learning the model parameters, we could estimate the baseline hazard with $\mathbf{h}_0(\cdot;\widehat{\theta})$ and the log partial hazard with $\mathbf{f}(\cdot;\widehat{\theta})$.

The rest of this section is organized as follows:

- (Section 3.1) First, assuming that the $\mathbf{h}_0(\cdot;\theta)$ and $\mathbf{f}(\cdot;\theta)$ are known (or alternatively that we have estimates for these), we show how the proportional hazards assumption restricts the shapes of $S(\cdot|x)$ that are possible across different $x \in \mathcal{X}$.
- (Section 3.2) Next, we go over the general procedure for learning $\mathbf{h}_0(\cdot;\theta)$ and $\mathbf{f}(\cdot;\theta)$ for parametric proportional hazards models as well as how to make predictions after model training. Our coverage here generalizes what we saw in Section 2 for the exponential and Weibull time-to-event prediction models.

- (Section 3.3) We then discuss *semiparametric* proportional hazards models, where $\mathbf{h}_0(\cdot;\theta)$ is learned nonparametrically while $\mathbf{f}(\cdot;\theta)$ is learned parametrically. Note that semiparametric proportional hazards models are commonly referred to as *Cox proportional hazards models* (which we just abbreviate as *Cox models*). The original Cox model [Cox, 1972] leaves $\mathbf{h}_0(\cdot;\theta)$ unspecified and assumes that $\mathbf{f}(\cdot;\theta) = \theta^\top x$, where $\theta \in \mathbb{R}^d$ and $x \in \mathcal{X} \subseteq \mathbb{R}^d$. The deep learning version (DeepSurv) [Faraggi and Simon, 1995, Katzman et al., 2018] also leaves $\mathbf{h}_0(\cdot;\theta)$ unspecified and replaces $\mathbf{f}(\cdot;\theta)$ with a neural network.
- (Section 3.4) Finally, we discuss an extension of the Cox model called Cox-Time [Kvamme et al., 2019] that removes the proportional hazards assumption.

A major selling point of the original Cox model [Cox, 1972] (where $\mathbf{f}(\cdot;\theta) = \theta^\top x$ with both θ and x belonging to $\mathbb{R}^d$) is that it is straightforward to interpret if the d input features themselves are interpretable (as is commonly the case for tabular data). In this setting, parameter vector $\theta = (\theta_1, \theta_2, \ldots, \theta_d)$ simply says how to weight each feature: the ℓ-th feature has weight θ_ℓ. In fact, a standard quantity used for model interpretation is called the *hazard ratio*, which is defined for the ℓ-th feature to be $\exp(\theta_\ell)$. The basic idea is as follows. Consider a feature vector $x \in \mathbb{R}^d$. Now consider a second feature vector $\widetilde{x} \in \mathbb{R}^d$ that is the same as x except that the ℓ-th entry is larger by 1, *i.e.*, $\widetilde{x}_j = x_j$ for every index $j \neq \ell$ whereas $\widetilde{x}_\ell = x_\ell + 1$. In this case, it turns out that the second feature vector $\widetilde{x}$ has a hazard value that is a multiplicative factor of $\exp(\theta_\ell)$ larger than that of x. To see this, note that

$$\begin{aligned}
\frac{h(t|\widetilde{x})}{h(t|x)} &= \frac{\cancel{\mathbf{h}_0(t;\theta)}\exp(\theta^\top \widetilde{x})}{\cancel{\mathbf{h}_0(t;\theta)}\exp(\theta^\top x)} \\
&= \frac{\exp\left(\theta_\ell(x_\ell + 1) + \sum_{j\neq\ell}\theta_j x_j\right)}{\exp\left(\sum_{j=1}^d \theta_j x_j\right)} \\
&= \frac{\exp\left(\theta_\ell + \sum_{j=1}^d \theta_j x_j\right)}{\exp\left(\sum_{j=1}^d \theta_j x_j\right)} \\
&= \exp(\theta_\ell).
\end{aligned}$$

To recap, under the proportional hazard assumption for the standard Cox model, an increase in the ℓ-th feature by 1 unit (with all other features being held the same) is associated with an increase in risk by a multiplicative factor given by the hazard ratio $\exp(\theta_\ell)$.

By replacing $\mathbf{f}(x;\theta) = \theta^\top x$ (a linear function of x) with a potentially highly nonlinear function, the DeepSurv model is more flexible than the original Cox model, but interpreting how the DeepSurv model makes predictions is less straightforward. In more detail, consider again the setting where raw inputs are feature vectors in $\mathbb{R}^d$. Reusing the earlier notation where $x \in \mathbb{R}^d$ and $\widetilde{x} \in \mathbb{R}^d$ differ only in the ℓ-th feature, with $\widetilde{x}_\ell = x_\ell + 1$, the hazard ratio would be

$$\frac{h(t|\widetilde{x})}{h(t|x)} = \frac{\cancel{\mathbf{h}_0(t;\theta)}\exp(\mathbf{f}(\widetilde{x};\theta))}{\cancel{\mathbf{h}_0(t;\theta)}\exp(\mathbf{f}(x;\theta))}.$$

When $\mathbf{f}$ is highly nonlinear, then this ratio does not, in general, simplify "nicely" and could depend on many parameters, unlike in the linear setting where the hazard ratio ends up depending only on a single parameter θ_ℓ. The Cox-Time model is even more flexible than the DeepSurv model and can be even less straightforward to interpret.

3.1 Constraint on Survival Function Shapes

In this section, we assume that we already know θ (or have an estimate for it), which means that we also know $\mathbf{h}_0(\cdot;\theta)$ and $\mathbf{f}(\cdot;\theta)$. Let's look at what the proportional hazards assumption implies about the shapes of survival functions that are possible. Recall from Summary 2.1 that by knowing

the hazard function $h(t|x)$, we can recover $H(t|x) = \int_0^t h(u|x)\mathrm{d}u$ and $S(t|x) = e^{-H(t|x)}$. Thus, under the proportional hazards assumption (equation (39)), the cumulative hazard function is

$$H(t|x) = \int_0^t h(u|x)\mathrm{d}u = \int_0^t \mathbf{h}_0(u;\theta)e^{\mathbf{f}(x;\theta)}\mathrm{d}u = e^{\mathbf{f}(x;\theta)} \underbrace{\int_0^t \mathbf{h}_0(u;\theta)\mathrm{d}u}_{=:\mathbf{H}_0(t;\theta)}, \tag{40}$$

where the newly defined $\mathbf{H}_0(\cdot;\theta)$ is referred to as the *baseline cumulative hazard function*. Then we recover the survival function

$$S(t|x) = e^{-H(t|x)} = e^{-e^{\mathbf{f}(x;\theta)}\mathbf{H}_0(t;\theta)} = [\underbrace{e^{-\mathbf{H}_0(t;\theta)}}_{=:\mathbf{S}_0(t;\theta)}]^{e^{\mathbf{f}(x;\theta)}}, \tag{41}$$

where we have the newly defined *baseline survival function* $\mathbf{S}_0(\cdot;\theta)$. Equation (41) tells us that all possible survival functions under the proportional hazards assumption must be powers of $\mathbf{S}_0(\cdot;\theta)$—see Figure 4(a) for an illustration. This is a strong assumption! *Survival functions that are not powers of $\mathbf{S}_0(\cdot;\theta)$ are impossible under the proportional hazards assumption (such as the green curve that crisscrosses the baseline survival function in Figure 4(b).*

As shown in Figure 4(a), the allowed survival functions that are closer to the origin—which have higher $\mathbf{f}(x;\theta)$ value—are uniformly worse than ones farther away from the origin, regardless of what time t we look at (as a reminder from Section 2.2.1, the area under a survival function is the mean survival time and the time at which the survival function crosses the y-axis value of $1/2$ is the median survival time). In particular, under a proportional hazards assumption, whether a data point with x is likely to have a shorter or longer survival time is entirely determined by the log partial hazard function value $\mathbf{f}(x;\theta)$ (which we previously pointed out could be interpreted as a risk score): higher values of $\mathbf{f}(x;\theta)$ correspond to shorter survival times.

3.2 Parametric Proportional Hazards Models

In Section 2.2.2, we had already shown how exponential and Weibull time-to-event prediction models could be learned via maximum likelihood and also how to subsequently make predictions. We now generalize both of these cases. Consider when $\mathbf{h}_0(\cdot;\theta)$ and $\mathbf{f}_0(\cdot;\theta)$ are parametric functions differentiable with respect to θ (where we assume that $\mathbf{h}_0(\cdot;\theta)$ is specified in continuous time just like in the exponential and Weibull examples), and we have some closed-form expression for

$$\mathbf{H}_0(t;\theta) := \int_0^t \mathbf{h}_0(u;\theta)\mathrm{d}u,$$

which is also differentiable with respect to θ.[15] Then learning θ and making predictions works similarly to what is described in Examples 2.2 and 2.3. We now state the general procedure.

Training. Model training amounts to learning the parameters θ, which we do by writing the loss function (that depends on training data $(X_1, Y_1, \Delta_1), \ldots, (X_n, Y_n, \Delta_n)$) and then optimizing:

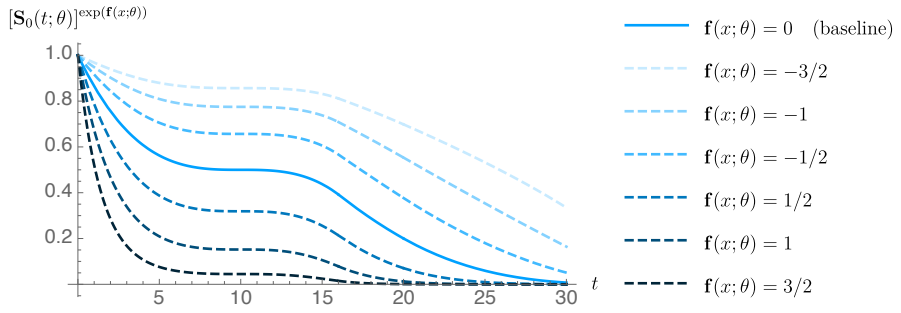
1. We write the negative log likelihood loss (equation (11)) but replace $\mathbf{h}(t|x;\theta)$ with

[15] For the exponential time-to-event prediction model, recall that $\theta = (\beta, \psi) \in \mathbb{R}^d \times \mathbb{R}$ and $\mathbf{h}_0(t;\theta) = e^{\psi}$, so that

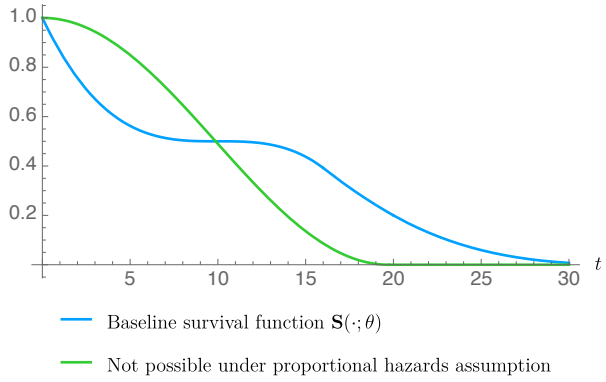
$$\mathbf{H}_0(t;\theta) = \int_0^t e^{\psi}\mathrm{d}u = te^{\psi}.$$

For the Weibull time-to-event prediction model, recall that $\theta = (\beta, \psi, \phi) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$ and $\mathbf{h}_0(t;\theta) = t^{e^{\phi}-1}e^{\psi+\phi}$, so that

$$\mathbf{H}_0(t;\theta) = \int_0^t u^{e^{\phi}-1}e^{\psi+\phi}\mathrm{d}u = t^{e^{\phi}}e^{\psi}.$$



(a)



(b)

Figure 4: Under the proportional hazards assumption (equation (39)), possible survival functions are all powers of the baseline survival function $\mathbf{S}_0(\cdot;\theta)$ as shown in panel (a); note that we can always unambiguously order these functions based on the log partial hazard function $\mathbf{f}(\cdot;\theta)$. In contrast, the green curve shown in panel (b) is not possible under a proportional hazards model and is neither uniformly better nor uniformly worse than the baseline survival function.

$\mathbf{h}_0(t;\theta)e^{\mathbf{f}(x;\theta)}$:

$$
\begin{aligned}
&\mathbf{L}_{\text{Hazard-NLL}}(\theta) \\
&= -\frac{1}{n}\sum_{i=1}^{n}\left\{\Delta_i \log \mathbf{h}(Y_i|X_i;\theta) - \int_0^{Y_i} \mathbf{h}(u|X_i;\theta)\mathrm{d}u\right\} \\
&= -\frac{1}{n}\sum_{i=1}^{n}\left\{\Delta_i \log(\mathbf{h}_0(Y_i;\theta)e^{\mathbf{f}(X_i;\theta)}) - \int_0^{Y_i} \mathbf{h}_0(u;\theta)e^{\mathbf{f}(X_i;\theta)}\mathrm{d}u\right\} \\
&= -\frac{1}{n}\sum_{i=1}^{n}\left\{\Delta_i[\mathbf{f}(X_i;\theta) + \log \mathbf{h}_0(Y_i;\theta)] - e^{\mathbf{f}(X_i;\theta)}\int_0^{Y_i} \mathbf{h}_0(u;\theta)\mathrm{d}u\right\} \\
&= -\frac{1}{n}\sum_{i=1}^{n}\left\{\Delta_i[\mathbf{f}(X_i;\theta) + \log \mathbf{h}_0(Y_i;\theta)] - e^{\mathbf{f}(X_i;\theta)}\mathbf{H}_0(Y_i;\theta)\right\}.
\end{aligned}
$$

2. We then use a standard neural network optimizer to solve

$$\widehat{\theta} := \arg\min_{\theta} \mathbf{L}_{\text{Hazard-NLL}}(\theta).$$

Prediction. To predict the hazard function, we simply use equation (39), where we plug in $\widehat{\theta}$ in place of θ:

$$\widehat{h}(t|x) := \mathbf{h}_0(t;\widehat{\theta})e^{\mathbf{f}(x;\widehat{\theta})} \quad \text{for } t \geq 0, x \in \mathcal{X}.$$

Predicting the cumulative hazard and survival functions then just amount to using the conversions from Summary 2.1. Specifically, we could estimate the cumulative hazard function using

$$\widehat{H}(t|x) := \int_0^t \widehat{h}(u|x)\mathrm{d}u = e^{\mathbf{f}(x;\widehat{\theta})} \int_0^t \mathbf{h}_0(u;\widehat{\theta})\mathrm{d}u = e^{\mathbf{f}(x;\widehat{\theta})}\mathbf{H}_0(t;\widehat{\theta}),$$

and the survival function using

$$\widehat{S}(t|x) := \exp(-\widehat{H}(t|x)) = \exp\big(-e^{\mathbf{f}(x;\widehat{\theta})}\mathbf{H}_0(t;\widehat{\theta})\big).$$

3.3 Semi-Parametric Proportional Hazards Models: DeepSurv

We now turn to the more complicated case where $\mathbf{h}_0(\cdot;\theta)$ is left unspecified (but we shall see that it still depends on parameter variable θ) while $\mathbf{f}(\cdot;\theta)$ remains parametric, resulting in a model commonly referred to as DeepSurv [Faraggi and Simon, 1995, Katzman et al., 2018]. We state the standard procedure for learning $\mathbf{f}(\cdot;\theta)$ and $\mathbf{h}_0(\cdot;\theta)$ followed by how the hazard, cumulative hazard, and survival functions are predicted. The training procedure does maximum likelihood estimation (for a derivation, see Section 3.A).

Training. Model training consists of two steps:

1. We estimate θ by numerically solving the optimization problem

$$\widehat{\theta} := \arg\min_{\theta} \frac{1}{n}\sum_{i=1}^{n} \Delta_i \bigg[-\mathbf{f}(X_i;\theta) + \log\bigg(\sum_{j=1}^{n} \mathbb{1}\{Y_j \geq Y_i\} e^{\mathbf{f}(X_j;\theta)}\bigg)\bigg] \tag{42}$$

   using a standard neural network optimizer.[16]

2. Next, we estimate $\mathbf{h}_0(\cdot;\theta)$ using the method by Breslow [1972]: consider the discrete time grid given by the unique times of death $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$. We define $\tau_{(0)} := 0$. Breslow assumes that $\mathbf{h}_0(\cdot;\theta)$ (defined in continuous time) is piecewise constant, where the possible changes in $\mathbf{h}_0(\cdot;\theta)$ happen at the discrete time grid points. Specifically, the Breslow estimator of $\mathbf{h}_0(\cdot;\theta)$ is

$$\widehat{\mathbf{h}}_0(t) := \begin{cases} \dfrac{D[\ell]}{(\tau_{(\ell)} - \tau_{(\ell-1)})\sum_{j=1}^{n} \mathbb{1}\{Y_j \geq \tau_{(\ell)}\} e^{\mathbf{f}(X_j;\widehat{\theta})}} & \text{if } \tau_{(\ell-1)} < t \leq \tau_{(\ell)} \\ & \quad \text{for } \ell \in [L], \\ 0 & \text{if } t > \tau_{(L)}. \end{cases} \tag{43}$$

   Note that $\widehat{\mathbf{h}}_0(\cdot)$ depends on $\widehat{\theta}$.

Prediction. To predict the hazard function $h(t|x) = \mathbf{h}_0(t;\theta)e^{\mathbf{f}(x;\theta)}$, we simply plug in the estimated $\widehat{\theta}$ and $\widehat{h}_0(\cdot)$:

$$\widehat{h}(t|x) := \widehat{\mathbf{h}}_0(t)e^{\mathbf{f}(x;\widehat{\theta})} \quad \text{for } t \geq 0, x \in \mathcal{X}.$$

To predict the cumulative hazard function, we use equation (40): $H(t|x) = e^{\mathbf{f}(x;\theta)}\mathbf{H}_0(t;\theta)$. In particular, we first estimate the baseline cumulative hazard with

$$\widehat{\mathbf{H}}_0(t) := \int_0^t \widehat{\mathbf{h}}_0(u)\mathrm{d}u = \sum_{m=1}^{L} \frac{\mathbb{1}\{\tau_{(m)} \leq t\} D[m]}{\sum_{j=1}^{n} \mathbb{1}\{Y_j \geq \tau_{(m)}\} e^{\mathbf{f}(X_j;\widehat{\theta})}} \quad \text{for } t \geq 0. \tag{44}$$

[16] In the classical setting where $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathbf{f}(x;\theta) = \theta^\top x$ with $\theta \in \mathbb{R}^d$, this problem is convex and can be solved using, for instance, Newton-Raphson. Moreover, in the classical setting, researchers have explored adding lasso [Tibshirani, 1997] or elastic net regularization [Zou and Hastie, 2005] on θ to encourage the estimated θ to, for instance, be sparse. In the neural network setting that we consider, standard tricks can be used for regularization, such as using dropout or weight decay.

Then we predict $H(\cdot|x)$ using the estimator

$$\widehat{\mathbf{H}}(t|x) := e^{\mathbf{f}(x;\widehat{\theta})}\widehat{\mathbf{H}}_0(t) \quad \text{for } t \geq 0, x \in \mathcal{X}. \tag{45}$$

Finally, to predict $S(\cdot|x)$ using the estimator

$$\widehat{S}(t|x) := e^{-H(t|x)} = \exp\big(-e^{\mathbf{f}(x;\widehat{\theta})}\widehat{\mathbf{H}}_0(t)\big) \quad \text{for } t \geq 0, x \in \mathcal{X}. \tag{46}$$

We provide a Jupyter notebook that shows how to implement DeepSurv.[17]

Remark 3.1 (Relationship to ranking)**.** The loss function in equation (42) is equal to

$$\frac{1}{n}\sum_{i=1}^{n}\Delta_i \log\left(\sum_{j=1}^{n}\mathbb{1}\{Y_j \geq Y_i\}e^{\mathbf{f}(X_j;\theta)-\mathbf{f}(X_i;\theta)}\right).$$

By carefully staring at this, notice the following. Minimizing this loss means that when $\Delta_i = 1$ and also $Y_i \leq Y_j$ (*i.e.*, the i-th point died and has an observed time before that of the j-th point), we would like risk score $\mathbf{f}(X_j;\theta)$ to be lower than risk score $\mathbf{f}(X_i;\theta)$. This directly corresponds to the intuition for the concordance index metric (Definition 2.1). In this sense, the Cox model's loss function could be interpreted as a ranking-based loss. For a more detailed explanation, see the paper by Raykar et al. [2007].

Remark 3.2 (The Nelson-Aalen estimator as a special case of the Cox model)**.** Breslow [1972] pointed out that in the special case where $\mathbf{f}(x;\theta) = 0$ for all $x \in \mathcal{X}$, then $\widehat{\mathbf{H}}_0(\cdot)$ from equation (44) actually just becomes the Nelson-Aalen estimator (equation (32)). In this case, the cumulative hazard estimate (equation (45)) would also just become the Nelson-Aalen estimator. However, the survival function estimator (equation (46)) would not be exactly equal to (but would approximate) the Kaplan-Meier estimator due to the result of Proposition 2.1.

Remark 3.3 (Piecewise constant functions in continuous time)**.** Note that $\widehat{h}_0(\cdot)$ in equation (43) could be thought of as a discrete time object since it only has L nonzero values. Thus, it could be stored on a computer in a 1D array/table. However, we want to emphasize that $\mathbf{h}_0(\cdot)$ really is a continuous time hazard function and *not* a forward-filled version of a discrete time hazard function like the ones we see in Section 2.3.

As a reminder, in Section 2.3, when we were working directly in discrete time, we required that the discrete time hazard function $h[\cdot|x]$ to have values that are at most 1 since they are probabilities. We used this to, for instance, derive that $S[\ell|x] = \prod_{m=1}^{\ell}(1-h[m|x])$.

In contrast, one can easily check that the Breslow estimator $\widehat{\mathbf{h}}_0(\cdot)$ could take on values that are arbitrarily large (possibly larger than 1) since $\mathbf{f}(\cdot;\theta)$ is unconstrained (so that the nonnegative term $e^{\mathbf{f}(X_j;\widehat{\theta})}$ could be arbitrarily large or small).

Later on in Section 5.2, we will show that we can convert any discrete time model like the ones in Section 2.3 to continuous time. However, what we are saying here is that for a hazard function that is defined originally in continuous time, even if it is piecewise constant, we cannot in general convert it to be a discrete time hazard function of the form we saw in Section 2.3.

[17] https://github.com/georgehc/survival-intro/blob/main/S3.3_DeepSurv.ipynb

3.4 Removing the Proportional Hazards Assumption: Cox-Time

A generalization of the Cox model called Cox-Time [Kvamme et al., 2019] replaces $\mathbf{f}(x;\theta)$ with the neural network $\mathbf{g}(x,t;\theta)$ that depends on both input $x \in \mathcal{X}$ and time $t \geq 0$. As a result, we have the factorization

$$h(t|x) = \mathbf{h}_0(t;\theta)e^{\mathbf{g}(x,t;\theta)} \quad \text{for } t \geq 0, x \in \mathcal{X}. \tag{47}$$

The earlier proportional hazards factorization from equation (39) has time t and input x contribute to different factors. Now, in equation (47), time t and input x could interact within the function $\mathbf{g}(x,t;\theta)$. Thus, whereas previously, the proportional hazards factorization constrained the survival function shapes to be powers of a baseline survival function (as discussed in Section 3.1), *this constraint is no longer guaranteed to hold in general for Cox-Time*.[18]

One might wonder why there needs to even be a baseline hazard function $\mathbf{h}_0(t;\theta)$ if $\mathbf{g}(x,t;\theta)$ already depends on time t. This has to do with how the Cox-Time model is trained. Kvamme et al. [2019] proposed simply using the DeepSurv training and prediction procedures with minor modifications to train the Cox-Time model.

Training. Model training consists of two steps:

1. We define the loss

$$\begin{aligned}&\mathbf{L}_{\text{Cox-Time}}(\theta)\\ &:= \frac{1}{n}\sum_{i=1}^{n}\Delta_i\bigg[-\mathbf{g}(X_i,Y_i;\theta) + \log\bigg(\sum_{j=1}^{n}\mathbb{1}\{Y_j \geq Y_i\}e^{\mathbf{g}(X_j,Y_i;\theta)}\bigg)\bigg].\end{aligned} \tag{48}$$

   Using a neural network optimizer, we compute $\widehat{\theta} := \arg\min_\theta \mathbf{L}_{\text{Cox-Time}}(\theta)$. The summation inside the log intentionally uses time Y_i and not Y_j (notice the expression "$\mathbf{g}(X_j,Y_i;\theta)$").

2. Denote the unique times of death by $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$, and define $\tau_{(0)} := 0$. We compute:

$$\begin{aligned}&\widehat{\mathbf{h}}_0(t)\\ &:= \begin{cases}\dfrac{D[\ell]}{(\tau_{(\ell)}-\tau_{(\ell-1)})\sum_{j=1}^{n}\mathbb{1}\{Y_j \geq \tau_{(\ell)}\}e^{\mathbf{g}(X_j,\tau_{(\ell)};\widehat{\theta})}} & \text{if } \tau_{(\ell-1)} < t \leq \tau_{(\ell)}\\ & \qquad \text{for } \ell \in [L],\\ 0 & \text{if } t > \tau_{(L)}.\end{cases}\end{aligned}$$

The reason why we still need to estimate the baseline hazard function $\mathbf{h}_0(\cdot;\theta)$ is as follows [Kvamme et al., 2019, Section 3.4]: if $\mathbf{g}(x,t;\theta) = \mathbf{a}(x,t;\theta) + \mathbf{b}(t;\theta)$ for some functions $\mathbf{a}(\cdot,\cdot;\theta)$ and $\mathbf{b}(\cdot;\theta)$, then the function $\mathbf{b}(\cdot;\theta)$ would actually be cancelled out in the loss $\mathbf{L}_{\text{Cox-Time}}(\theta)$. Thus, minimizing $\mathbf{L}_{\text{Cox-Time}}(\theta)$ would not be able to learn $\mathbf{b}(\cdot;\theta)$. Instead, we learn $\mathbf{b}(\cdot;\theta)$ as part of the baseline hazard function.

Prediction. We predict the hazard function $h(\cdot|x)$ using

$$\widehat{h}(t|x) := \widehat{\mathbf{h}}_0(t)e^{\mathbf{g}(x,t;\widehat{\theta})} \quad \text{for } t \geq 0, x \in \mathcal{X}.$$

To predict the cumulative hazard function, we first compute

$$\widehat{\mathbf{H}}_0(t) := \int_0^t \widehat{\mathbf{h}}_0(u)\mathrm{d}u = \sum_{m=1}^{L}\frac{\mathbb{1}\{\tau_{(m)} \leq t\}D[m]}{\sum_{j=1}^{n}\mathbb{1}\{Y_j \geq \tau_{(m)}\}e^{\mathbf{g}(X_i,\tau_{(m)};\widehat{\theta})}} \quad \text{for } t \geq 0.$$

[18] We point out that the idea of replacing $\mathbf{f}(x;\theta)$ with a function that depends on both raw input x and time t is an innovation that predates the paper by Kvamme et al. [2019]. For example, Chapter 9 of the textbook by Klein and Moeschberger [2003] suggests setting $\mathbf{f}$ to still have a linear form but where there are some newly added features that capture interactions between some of the original features and a pre-specified function of time (*e.g.*, see equation (9.2.2) of Klein and Moeschberger [2003]). It is possible to specify such an $\mathbf{f}$ that still enables statistical inference (see Example 9.2 of Klein and Moeschberger [2003]). Cox-Time was not designed with statistical inference in mind and aims to simply replace $\mathbf{f}$ with an arbitrarily complex neural network that depends on both x and t, with the hope that after model training, the neural network will encode interactions between x and t that are relevant for prediction.

Then we predict $H(\cdot|x)$ using

$$\widehat{H}(t|x) := e^{\mathbf{g}(x,t;\widehat{\theta})}\widehat{\mathbf{H}}_0(t) \quad \text{for } t \geq 0, x \in \mathcal{X}.$$

We predict the survival function $S(\cdot|x)$ using

$$\widehat{S}(t|x) := e^{-H(t|x)} = \exp\big(-e^{\mathbf{g}(x,t;\widehat{\theta})}\widehat{\mathbf{H}}_0(t)\big) \quad \text{for } t \geq 0, x \in \mathcal{X}.$$

Overall, the training and prediction procedures for Cox-Time are heuristically justified. It is unclear whether there is a more theoretically sound manner for jointly estimating $\mathbf{h}_0(\cdot;\theta)$ and $\mathbf{g}(\cdot,\cdot;\theta)$, where $\mathbf{h}_0(\cdot;\theta)$ is left unspecified and $\mathbf{g}(\cdot,\cdot;\theta)$ is within, say, some reasonably wide family of neural networks.

In our companion code repository, we provide a Jupyter notebook that implements Cox-Time.[19] Our notebook uses the original Cox-Time code by Kvamme et al. [2019] for which the loss $\mathbf{L}_{\text{Cox-Time}}(\theta)$ in equation (48) is *not* computed exactly. To speed up computation, in equation (48), the summation (over index j) inside the log is approximated by randomly sampling one of the nonzero terms of the summation. This is referred to as a "case-control" approximation, with the idea that for each training point i (the outer summation of equation (48)) that we call the "case" data point, we are randomly choosing a single other training point (called the "control") to be used inside the log.

3.A Technical Details: Derivation of the Cox Model's Two-Step Maximum Likelihood Estimator

The derivation we present here spells out the steps of the terse derivation given by Breslow [1972] that was for the original Cox model (specifically, the first few paragraphs of his discussion outlines how the derivation works). Breslow's derivation trivially can be adapted to the DeepSurv setting, where the log partial hazard function is specified by a neural network and is not simply a linear function.

As a reminder, we discretize time so that $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$ are the unique times of death, and we set $\tau_{(0)} := 0$. For a noncensored data point i, let $\kappa(Y_i) \in [L]$ denote the time index corresponding to time Y_i. Extremely importantly, as Breslow [1972] states in his derivation, he follows Kalbfleisch and Prentice's convention and takes a censored data point's observed time to be the *preceding* observed time of death. In other words, for a censored data point i, we take $\kappa(Y_i) \in [L]$ to be the time index of the largest time of death that is before Y_i (if there is no death prior to Y_i, then we take $\kappa(Y_i) = 0$).

Next, suppose that $\mathbf{h}_0(t;\theta)$ is piecewise constant so that

$$\mathbf{h}_0(t;\theta) := \begin{cases} \lambda_\ell & \text{if } \tau_{(\ell-1)} < t \leq \tau_{(\ell)} \text{ for } \ell \in [L], \\ 0 & \text{if } t > \tau_{(L)}, \end{cases}$$

where $\lambda := (\lambda_1, \lambda_2, \ldots, \lambda_L) \in [0,\infty)^L$. Thus, the hazard function is equal to

$$\mathbf{h}(t|x;\theta) := \mathbf{h}_0(t;\theta)e^{\mathbf{f}(x;\theta)}.$$

The hazard form of the log likelihood (equation (10)) is (where we emphasize both the dependence on θ and λ):

$$\begin{aligned} &\log \mathcal{L}(\theta,\lambda) \\ &= \sum_{i=1}^{n} \left\{ \Delta_i \log \mathbf{h}(Y_i|X_i;\theta) - \int_0^{Y_i} \mathbf{h}(u|X_i;\theta)\mathrm{d}u \right\} \\ &= \sum_{i=1}^{n} \left\{ \Delta_i \log(\mathbf{h}_0(Y_i;\theta)e^{\mathbf{f}(X_i;\theta)}) - \int_0^{Y_i} \mathbf{h}_0(u;\theta)e^{\mathbf{f}(X_i;\theta)}\mathrm{d}u \right\} \end{aligned}$$

[19] https://github.com/georgehc/survival-intro/blob/main/S3.4_Cox-Time.ipynb

$$
\begin{aligned}
&= \sum_{i=1}^{n} \left\{ \Delta_i \log(\mathbf{h}_0(Y_i;\theta) e^{\mathbf{f}(X_i;\theta)}) - e^{\mathbf{f}(X_i;\theta)} \int_0^{Y_i} \mathbf{h}_0(u;\theta) \mathrm{d}u \right\} \\
&= \sum_{i=1}^{n} \left\{ \Delta_i \log(\lambda_{\kappa(Y_i)} e^{\mathbf{f}(X_i;\theta)}) - e^{\mathbf{f}(X_i;\theta)} \sum_{m=1}^{\kappa(Y_i)} (\tau_{(m)} - \tau_{(m-1)}) \lambda_m \right\} \\
&= \sum_{i=1}^{n} \left\{ \Delta_i \log \lambda_{\kappa(Y_i)} + \Delta_i \mathbf{f}(X_i;\theta) - e^{\mathbf{f}(X_i;\theta)} \sum_{m=1}^{\kappa(Y_i)} (\tau_{(m)} - \tau_{(m-1)}) \lambda_m \right\} \\
&= \sum_{i=1}^{n} \Delta_i \log \lambda_{\kappa(Y_i)} + \sum_{i=1}^{n} \Delta_i \mathbf{f}(X_i;\theta) - \sum_{i=1}^{n} e^{\mathbf{f}(X_i;\theta)} \sum_{m=1}^{\kappa(Y_i)} (\tau_{(m)} - \tau_{(m-1)}) \lambda_m \\
&= \sum_{m=1}^{L} D[m] \log \lambda_{(m)} + \sum_{i=1}^{n} \Delta_i \mathbf{f}(X_i;\theta) \\
&\quad - \sum_{m=1}^{L} (\tau_{(m)} - \tau_{(m-1)}) \lambda_m \sum_{j=1}^{n} \mathbb{1}\{Y_j \geq m\} e^{\mathbf{f}(X_j;\theta)}.
\end{aligned} \tag{49}
$$

Setting the derivative with respect to $\lambda_{(\ell)}$ to 0, we get

$$
0 = \Big[\frac{\mathrm{d} \log \mathcal{L}(\theta)}{\mathrm{d}\lambda_{(\ell)}} \Big]_{\lambda_{(\ell)} = \widehat{\lambda}_{(\ell)}} = \frac{D[\ell]}{\widehat{\lambda}_{(\ell)}} - (\tau_{(\ell)} - \tau_{(\ell-1)}) \sum_{j=1}^{n} \mathbb{1}\{Y_j \geq \ell\} e^{\mathbf{f}(X_j;\theta)}.
$$

Rearranging terms, we get

$$
\widehat{\lambda}_{(\ell)} = \frac{D[\ell]}{(\tau_{(\ell)} - \tau_{(\ell-1)}) \sum_{j=1}^{n} \mathbb{1}\{Y_j \geq \ell\} e^{\mathbf{f}(X_j;\theta)}}, \tag{50}
$$

which is strictly positive. One can verify that $\big[\frac{\mathrm{d}^2 \log \mathcal{L}(\theta)}{\mathrm{d}\lambda_{(\ell)}^2} \big]_{\lambda_{(\ell)} = \widehat{\lambda}_{(\ell)}} < 0$ so that indeed we are looking at a maximum. Plugging in the optimal choice of $\widehat{\lambda}_{(\ell)}$ back into equation (49), we get

$$
\begin{aligned}
&\log \mathcal{L}(\theta, \widehat{\lambda}) \\
&\quad = \sum_{m=1}^{L} D[m] \log \frac{D[\ell]}{(\tau_{(m)} - \tau_{(m-1)}) \sum_{j=1}^{n} \mathbb{1}\{Y_j \geq m\} e^{\mathbf{f}(X_j;\theta)}} \\
&\qquad + \sum_{i=1}^{n} \Delta_i \mathbf{f}(X_i;\theta) - \sum_{m=1}^{L} D[m] \\
&\quad = \sum_{i=1}^{n} \Delta_i \mathbf{f}(X_i;\theta) - \sum_{m=1}^{L} D[m] \log \sum_{j=1}^{n} \mathbb{1}\{Y_j \geq m\} e^{\mathbf{f}(X_j;\theta)} \\
&\qquad + \underbrace{\sum_{m=1}^{L} D[m] \log \frac{D[\ell]}{(\tau_{(m)} - \tau_{(m-1)})} - \sum_{m=1}^{L} D[m]}_{\text{constant (with respect to } \theta)} \\
&\quad = \sum_{i=1}^{n} \Delta_i \mathbf{f}(X_i;\theta) - \sum_{i=1}^{n} \Delta_i \log \sum_{j=1}^{n} \mathbb{1}\{Y_j \geq Y_i\} e^{\mathbf{f}(X_j;\theta)} + \text{constant} \\
&\quad = \sum_{i=1}^{n} \Delta_i \Big[\mathbf{f}(X_i;\theta) - \sum_{i=1}^{n} \log \sum_{j=1}^{n} \mathbb{1}\{Y_j \geq Y_i\} e^{\mathbf{f}(X_j;\theta)} \Big] + \text{constant}.
\end{aligned}
$$

Then

$$\begin{aligned}\widehat{\theta} &:= \arg\max_{\theta} \log \mathcal{L}(\theta, \widehat{\lambda}) \\ &= \arg\min_{\theta} -\frac{1}{n} \log \mathcal{L}(\theta, \widehat{\lambda}) \\ &= \arg\min_{\theta} -\frac{1}{n} \sum_{i=1}^{n} \Delta_i \left[\mathbf{f}(X_i; \theta) - \sum_{i=1}^{n} \log \sum_{j=1}^{n} \mathbb{1}\{Y_j \geq Y_i\} e^{\mathbf{f}(X_j;\theta)} \right],\end{aligned}$$

where we have dropped the constant as it does not affect the argument achieving the minimum. This finishes the derivation of the first step of the Cox training procedure (namely, equation (42)). The second step of the Cox training procedure simply plugs in the optimal choice of $\widehat{\theta}$ in place of θ in equation (50).

4 Deep Conditional Kaplan-Meier Estimators

In Section 2, we encountered the Kaplan-Meier estimator [Kaplan and Meier, 1958], which estimates the population-level survival function $S_{\text{pop}}(t) := \mathbb{P}(T > t)$. In this section, we describe a wide class of deep learning variants of the Kaplan-Meier estimator that estimate survival function $S(\cdot|x)$.[20] Specifically, we cover what are called *deep kernel survival analysis* (DKSA) models [Chen, 2020, 2024]. As we shall see, these models provide a couple different notions of interpretability. A special case of these models also has a theoretical accuracy guarantee. In experiments on standard datasets, these models have been shown to be competitive with various deep time-to-event prediction models.

As a reminder, classically, the Kaplan-Meier estimator discretizes time using the unique times of death $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$. At each time index $\ell \in [L]$, we keep track of the number of deaths at time index ℓ (denoted as $D[\ell]$) and the number of points at risk at time index ℓ (denoted as $N[\ell]$):

$$D[\ell] := \sum_{j=1}^{n} \mathbb{1}\{Y_j = \tau_{(\ell)}\} \Delta_j, \qquad \text{(29, partially reproduced)}$$

$$N[\ell] := \sum_{j=1}^{n} \mathbb{1}\{Y_j \geq \tau_{(\ell)}\}. \qquad \text{(30, partially reproduced)}$$

Then the Kaplan-Meier estimator is given by

$$\widehat{S}_{\text{KM}}(t) := \prod_{m=1}^{L} \left(1 - \frac{D[m]}{N[m]}\right)^{\mathbb{1}\{\tau_{(m)} \leq t\}} \quad \text{for } t \geq 0. \qquad \text{(33, partially reproduced)}$$

We start from this classical nonparametric estimator and build our way to increasingly more sophisticated methods:

- (Section 4.1) We begin by presenting conditional Kaplan-Meier estimators [Beran, 1981], which estimate $S(\cdot|x)$ instead of $S_{\text{pop}}(\cdot)$. Conditional Kaplan-Meier estimators build on a simple idea: to predict a survival function specific to $x \in \mathcal{X}$, first determine which of the training raw inputs $X_1, \ldots, X_n$ are the k closest ones to x. We then compute a Kaplan-Meier survival function using only these k training points' ground truth labels. In this manner, we just constructed a survival function that depends on x. Moreover, any prediction comes with "evidence" as we would know exactly which k training points contributed to the prediction.

  A more elaborate "kernel" version is to weight the contribution of each training point based on how similar it is to x. Specifically, the user pre-specifies a kernel function (also called a

[20] Backing out estimates of the hazard and cumulative hazard functions is also possible, but for simplicity, we focus just on predicting $S(\cdot|x)$ in this section.

similarity function) $K : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ that measures how similar any two inputs $x, x' \in \mathcal{X}$ are (*e.g.*, $K(x, x') := \exp(-\|x - x'\|^2)$). We explain how this so-called *kernel Kaplan-Meier estimator* works.

- (Section 4.2) The problem with the kernel Kaplan-Meier estimator is that how well it works in practice depends heavily on the kernel function used. To address this problem, we present the DKSA framework by Chen [2020] that automatically learns the kernel function in a neural network framework. Prediction is still done exactly the same way as the kernel Kaplan-Meier estimator, just with an automatically learned kernel function.
- (Section 4.3) The kernel Kaplan-Meier estimator is computationally expensive for large datasets. When making predictions, a naive implementation would need to compute a similarity score (using the kernel function) of each test raw input x with every training input X_i. We go over a compression strategy by Chen [2024] that results in a class of models called *survival kernets*. Conceptually, survival kernets could be viewed as representing any data point as a combination of a few clusters, each of which could be visualized in terms of how it relates to raw input features and also to time-to-event outcomes. A special case of survival kernets has a theoretical accuracy guarantee.

4.1 Conditional Kaplan-Meier Estimators: k Nearest Neighbor and Kernel Variants

The basic idea of conditional Kaplan-Meier estimators [Beran, 1981] is that we could compute the Kaplan-Meier estimator restricted to (or "conditioned on") using only a subset of our training dataset and not necessarily all of it. Beran suggested both k nearest neighbor and kernel Kaplan-Meier estimators.

k nearest neighbor Kaplan-Meier estimator. How the k nearest neighbor Kaplan-Meier estimator works is that we first specify a distance function $\rho : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ between any two raw inputs (for example, if $\mathcal{X} = \mathbb{R}^d$, then ρ could be Euclidean distance). The distance between test raw input x and training raw input X_i is thus $\rho(x, X_i)$. Let $(X_{(1)}, Y_{(1)}, \Delta_{(1)}), (X_{(2)}, Y_{(2)}, \Delta_{(2)}), \ldots, (X_{(n)}, Y_{(n)}, \Delta_{(n)})$ denote the training raw inputs sorted so that $X_{(1)}$ is the closest training raw input to x (according to distance function ρ), $X_{(2)}$ is the second closest, and so forth. Then the k nearest neighbor Kaplan-Meier estimator simply computes the standard Kaplan-Meier estimator only using the k ground truth outcome labels $(Y_{(1)}, \Delta_{(1)}), (Y_{(2)}, \Delta_{(2)}) \ldots, (Y_{(k)}, \Delta_{(k)})$. We recover the standard Kaplan-Meier estimator by choosing $k = n$.

Kernel Kaplan-Meier estimator. The more general kernel version that Beran suggested assigns weights to different training points according to kernel function $K : \mathcal{X} \times \mathcal{X} \to [0, \infty)$, where $K(x, X_i)$ having a higher value means that x and X_i are "more similar". Then we generalize the definitions for the number of deaths $D[\ell]$ and number of data points at risk $N[\ell]$ at time $\tau_{(\ell)}$ from equations (29) and (30), respectively, into kernel versions. Specifically, for $\ell \in [L]$ and $x \in \mathcal{X}$, we define

$$D_{\text{kernel}}[\ell|x] := \sum_{j=1}^{n} K(x, X_j) \mathbb{1}\{Y_j = \tau_{(\ell)}\} \Delta_j,$$

$$N_{\text{kernel}}[\ell|x] := \sum_{j=1}^{n} K(x, X_j) \mathbb{1}\{Y_j \geq \tau_{(\ell)}\}.$$

Here, $D_{\text{kernel}}[\ell|x]$ could be thought of as the number of deaths at time $\tau_{(\ell)}$ among training points who "look like" x, where we have weighted each training point $j \in [n]$ by its similarity to x (the nonnegative weight $K(x, X_j)$). Likewise, $N_{\text{kernel}}[\ell|x]$ could be thought of as the number of training points at risk at time $\tau_{(\ell)}$ among those who "look like" x.

Then the kernel Kaplan-Meier estimator is given by

$$\widehat{S}_{\text{kernel-KM}}(t|x) := \prod_{m=1}^{L} \left(1 - \frac{D_{\text{kernel}}[m|x]}{N_{\text{kernel}}[m|x]}\right)^{\mathbb{1}\{\tau_{(m)} \le t\}} \quad \text{for } t \ge 0, x \in \mathcal{X},$$

with the convention that in the product, we ignore any time index m such that $N_{\text{kernel}}[m|x] = 0$, and if the product is "empty" (there are no terms to multiply), then the output is just 1.

The standard Kaplan-Meier estimator could be obtained by just setting $K(x, x') = 1$ for all $x, x' \in \mathcal{X}$. The k-nearest neighbor Kaplan-Meier estimator corresponds to the case where $K(x, X_j) = 1$ if X_j is one of the k nearest neighbors of x (among training raw inputs $X_1, X_2, \ldots, X_n$) according to some distance function $\rho : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ (such as Euclidean distance), and $K(x, X_j) = 0$ otherwise.

Similar to how the Kaplan-Meier estimator can be viewed as having a corresponding discrete time hazard function (equation (31)), so too does the kernel Kaplan-Meier estimator. Specifically, we can define the kernel hazard function estimate to be

$$\widehat{h}_{\text{kernel}}[\ell|x] := \frac{D_{\text{kernel}}[\ell|x]}{N_{\text{kernel}}[\ell|x]} = \frac{\sum_{j=1}^{n} K(x, X_j) \mathbb{1}\{Y_j = \tau_{(\ell)}\} \Delta_j}{\sum_{j=1}^{n} K(x, X_j) \mathbb{1}\{Y_j \ge \tau_{(\ell)}\}}, \tag{51}$$

which the convention that if the denominator is 0, then we just output $\widehat{h}_{\text{kernel}}[\ell|x] = 0$. In particular,

$$\widehat{S}_{\text{kernel-KM}}(t|x) = \prod_{m=1}^{L} (1 - \widehat{h}_{\text{kernel}}[\ell|x])^{\mathbb{1}\{\tau_{(m)} \le t\}} \quad \text{for } t \ge 0, x \in \mathcal{X}.$$

Interpreting predictions. An appealing aspect of the kernel Kaplan-Meier estimator is that when predicting $S(\cdot|x)$, we know how much any training point $j \in [n]$ contributes to the prediction since it is just given by the similarity score $K(x, X_j)$. Thus, in some sense, we have "evidence" for every prediction made, as we could always look at which training points contributed the most to the prediction. How interpretable this evidence is depends on whether it is straightforward for a user to make sense of these "most similar" training points to x.

Theoretical guarantees. Meanwhile, we point out that theory for k nearest neighbor and Kaplan Meier estimators is well-understood. Much like how as $n \to \infty$, the Kaplan-Meier estimator, under fairly general settings, converges to $S_{\text{pop}}(t) := \mathbb{P}(T > t)$ for all times t that are not too large [Földes and Rejtö, 1981], a similar result holds for the k nearest neighbor and kernel Kaplan-Meier estimators provably converging to $S(t|x)$ for all times t that are not too large (Chen [2019] provides rates of convergence in the case where raw inputs could reside in separable metric spaces, of which Euclidean space is a special case).

4.2 Learning the Kernel Function: Deep Kernel Survival Analysis

In practice, how well the kernel Kaplan-Meier estimator works heavily depends on the choice of the kernel function K [Chen, 2019]. To automatically learn K, Chen [2020] proposed an approach called deep kernel survival analysis (DKSA). Specifically, Chen set K equal to

$$\mathbf{K}(x, x'; \theta) := \exp(-\|\mathbf{f}(x; \theta) - \mathbf{f}(x'; \theta)\|^2) \quad \text{for } x, x' \in \mathcal{X}, \tag{52}$$

where $\mathbf{f}(\cdot; \theta)$ is a neural network with parameter variable θ that maps from the raw input space $\mathcal{X}$ to a latent embedding space $\mathbb{R}^{d_{\text{emb}}}$ for a user-specified embedding space dimension d_{emb}.[21]

We now state how prediction works as it is fairly straightforward and, moreover, the training procedure (that learns θ) depends on the prediction procedure.

[21] Traditionally, the kernel Kaplan-Meier estimator would be used in a simplistic manner where the user specifies $\mathbf{K}(x, x'; \theta)$ in terms of a single real-valued "bandwidth" parameter. For instance, we could have $\mathbf{f}(x; \theta) = \frac{x}{\sqrt{2\sigma^2}}$ (with $\theta = \sigma$) so that $\mathbf{K}(\cdot, \cdot; \theta)$ is a Gaussian kernel with standard deviation parameter σ (that is taken to be the "bandwidth"). Other ways to parameterize $\mathbf{K}(x, x'; \theta)$ in terms of $\mathbf{f}(\cdot; \theta)$ are possible, *e.g.*, $\mathbf{K}(x, x'; \theta) = 1/[e^{\|\mathbf{f}(x;\theta) - \mathbf{f}(x';\theta)\|} + 2 + e^{-\|\mathbf{f}(x;\theta) - \mathbf{f}(x';\theta)\|}]$.

Prediction. We use the kernel Kaplan-Meier estimator with the learned kernel function from equation (52). Specifically, we set $\widehat{h}_{\text{kernel}}[\ell|x]$ from equation (51) equal to

$$\begin{aligned}\mathbf{h}_{\text{DKSA}}[\ell|x;\theta] &:= \frac{\sum_{j=1}^n \mathbf{K}(x, X_j;\theta)\mathbb{1}\{Y_j = \tau_{(\ell)}\}\Delta_j}{\sum_{j=1}^n \mathbf{K}(x, X_j;\theta)\mathbb{1}\{Y_j \geq \tau_{(\ell)}\}} \\ &= \frac{\sum_{j=1}^n \exp(-\|\mathbf{f}(x;\theta) - \mathbf{f}(X_j;\theta)\|^2)\mathbb{1}\{Y_j = \tau_{(\ell)}\}\Delta_j}{\sum_{j=1}^n \exp(-\|\mathbf{f}(x;\theta) - \mathbf{f}(X_j;\theta)\|^2)\mathbb{1}\{Y_j \geq \tau_{(\ell)}\}},\end{aligned} \tag{53}$$

where, in terms of notation, on the left-hand side, we have dropped the hat "$\widehat{\ }$" over the function name to emphasize that at this point, the function has not actually been estimated yet since we still need to learn θ. Meanwhile, the survival function is given by

$$\mathbf{S}_{\text{DKSA}}(t|x;\theta) := \prod_{m=1}^{L} (1 - \mathbf{h}_{\text{DKSA}}[\ell|x;\theta])^{\mathbb{1}\{\tau_{(m)} \leq t\}} \quad \text{for } t \geq 0, x \in \mathcal{X}.$$

If we can come up with an estimate $\widehat{\theta}$ of θ, then we could then predict $S(\cdot|x)$ using

$$\widehat{S}_{\text{DKSA}}(t|x) := \mathbf{S}_{\text{DKSA}}(t|x;\widehat{\theta}) \quad \text{for } t \geq 0, x \in \mathcal{X}. \tag{54}$$

Training. The basic idea of how to learn θ is to plug in the hazard function estimate (equation (53)) into the hazard form of the discrete time negative log likelihood loss (equation (24)), *i.e.*, we could use the loss function

$$\begin{aligned}\mathbf{L}_{\text{DKSA-NLL-naive}}(\theta) := -\frac{1}{n}\sum_{i=1}^n \Bigg\{ &\Delta_i \log(\mathbf{h}_{\text{DKSA}}[\kappa(Y_i)|X_i;\theta]) \\ &+ (1-\Delta_i)\log(1 - \mathbf{h}_{\text{DKSA}}[\kappa(Y_i)|X_i;\theta]) \\ &+ \sum_{m=1}^{\kappa(Y_i)-1} \log(1 - \mathbf{h}_{\text{DKSA}}[m|X_i;\theta]) \Bigg\},\end{aligned}$$

where, as a reminder, $\kappa(Y_i) \in [L]$ is the time index that Y_i corresponds to. However, it turns out that minimizing $\mathbf{L}_{\text{DKSA-NLL-naive}}(\theta)$ works poorly in practice due to overfitting issues. Specifically note that

$$\mathbf{h}_{\text{DKSA}}[\ell|X_i;\theta] = \frac{\sum_{j=1}^n \exp(-\|\mathbf{f}(X_i;\theta) - \mathbf{f}(X_j;\theta)\|^2)\mathbb{1}\{Y_j = \tau_{(\ell)}\}\Delta_j}{\sum_{j=1}^n \exp(-\|\mathbf{f}(X_i;\theta) - \mathbf{f}(X_j;\theta)\|^2)\mathbb{1}\{Y_j \geq \tau_{(\ell)}\}},$$

which means that to predict the hazard function for X_i, on the right-hand side, we use ground truth outcome labels (namely Y_i and Δ_i) from the i-th training point itself. Thus, Chen [2020] suggested instead using a leave-one-out hazard estimator during model training:

$$\begin{aligned}\mathbf{h}_{\text{DKSA-train}}[\ell|i;\theta] := \frac{\sum_{j\neq i} \exp(-\|\mathbf{f}(X_i;\theta) - \mathbf{f}(X_j;\theta)\|^2)\mathbb{1}\{Y_j = \tau_{(\ell)}\}\Delta_j}{\sum_{j\neq i} \exp(-\|\mathbf{f}(X_i;\theta) - \mathbf{f}(X_j;\theta)\|^2)\mathbb{1}\{Y_j \geq \tau_{(\ell)}\}} \\ \text{for } \ell \in [L], i \in [n],\end{aligned} \tag{55}$$

which predicts the hazard function for the i-th training point only using the other training points' ground truth information. On the left-hand side, our notation now intentionally makes it clear that $\mathbf{h}_{\text{DKSA-train}}[\ell|i;\theta]$ is not meant to be evaluated at an arbitrary raw input x. Instead, it is only used to predict the hazard function for each training point $i \in [n]$.

The final negative log likelihood training loss used by Chen [2020] is

$$\mathbf{L}_{\text{DKSA-NLL}}(\theta) := -\frac{1}{n}\sum_{i=1}^{n}\left\{\Delta_i \log(\mathbf{h}_{\text{DKSA-train}}[\kappa(Y_i)|i;\theta]) \right.$$
$$+ (1-\Delta_i)\log(1-\mathbf{h}_{\text{DKSA-train}}[\kappa(Y_i)|i;\theta])$$
$$\left. + \sum_{m=1}^{\kappa(Y_i)-1} \log(1-\mathbf{h}_{\text{DKSA-train}}[m|i;\theta])\right\}.$$

This loss could be numerically minimized using a neural network optimizer to estimate $\widehat{\theta} := \arg\min_\theta \mathbf{L}_{\text{DKSA-NLL}}(\theta)$. We emphasize here that using a variant of minibatch gradient descent is important rather than using full batch gradient descent (that uses the entire training data per optimization step). In particular, the loss function would require computation time that scales as $\mathcal{O}(n^2)$ when using full batch gradient descent. By using minibatches of batch size b that the user specifies (where $b < n$), the loss function would only be computed for training data in each minibatch, with computation time scaling as $\mathcal{O}(b^2)$.

We mention several refinements that are important in practice to getting DKSA to work well:

- When coming up with an initial guess for θ during neural network optimizer, standard random parameter initialization (*e.g.*, He et al. 2015) tends to work poorly compared to a random initialization strategy based on tree ensembles [Chen, 2020, 2024]. In particular, Chen [2024, Section 4] showed how to randomly initialize θ with the help of XGBoost; this initialization strategy can scale to large datasets. The rough idea is that an already trained XGBoost model comes with a kernel function $\widetilde{K} : \mathcal{X} \times \mathcal{X} \to [0,1]$ that we could easily evaluate for any pair of inputs [Chen and Shah, 2018, Section 7.1.3]. We initialize θ by first minimizing (in minibatches) a mean squared error loss between $\mathbf{K}(\cdot,\cdot;\theta)$ and $\widetilde{K}$ (evaluated on data for a single minibatch at a time) for some number of iterations.

- How time is discretized during training and how time is interpolated for prediction matter in practice [Chen, 2020, 2024]. For time discretization, for some datasets, it could be helpful to use all unique times of death whereas for other datasets, using a coarser grid is helpful (which requires some preprocessing that discretizes the Y_i values seen in training data into some user-specified number of bins; we gave examples of ways to do this in Section 2.3.2). As for time interpolation, even though for simplicity we have presented the kernel Kaplan-Meier estimator using forward filling interpolation, in practice, using a more sophisticated interpolation strategy such as constant density or constant hazard interpolation is better (an explanation of how these work is provided by Kvamme and Borgan [2021]). In short, we advise against using forward filling interpolation.

- Chen [2020, Section 6] discussed how to incorporate the DeepHit model's ranking loss [Lee et al., 2018] although he did not implement it in the original DKSA paper. The subsequent work by Chen [2024] does include this ranking loss (so that the overall training loss is the sum of the negative log likelihood loss $\mathbf{L}_{\text{DKSA-NLL}}(\theta)$ and the ranking loss), which improves the achieved time-dependent concordance index of the resulting model.

These refinements, however, do not remedy the issue that after model training, when we predict $S(\cdot|x)$ for a single test raw input $x \in \mathcal{X}$, computing the prediction $\widehat{S}_{\text{DKSA}}(\cdot|x)$ using equation (54) would involve computing $\mathbf{h}_{\text{DKSA}}[\ell|x;\widehat{\theta}]$ using equation (53), which iterates through all n training points. When n is large, this computation is impractical. We resolve this issue in the next section. The resolution turns out to also help with model interpretability and comes with an accuracy guarantee.

4.3 Scalable Deep Kernel Survival Analysis: Survival Kernets

To accelerate prediction for a single raw input, we want to avoid having to look at all n training points. Chen [2024] proposed a method for doing this that uses two key ideas:

- First, after training a DKSA model (which can scale to large datasets using the ideas we presented in the previous section), we cluster the training points using an exemplar-based clustering method. In other words, each cluster has an "exemplar" that is an actual training point that represents the cluster. When we make predictions, we only compute the similarity between test raw input x and these exemplars. This idea could be thought of as us compressing training data into a few clusters.

  There will be a hyperparameter $\varepsilon \geq 0$ that controls how much compression happens, where $\varepsilon = 0$ means that there is no compression (in which case prediction will be slower) whereas when $\varepsilon \to \infty$, then we maximally compress the data (so that there's only a single cluster), but this will mean that we predict the exact same survival function regardless of what the test raw input x is.

- Second, we ignore any exemplar that is "too far" from x. A key idea here is to exploit existing fast nearest neighbor search algorithms that enable us to quickly find only the exemplars that are close enough to x.

  There will be a distance threshold hyperparameter $\tau > 0$ that controls what it means to be too far. Having $\tau \to \infty$ will mean that there is no distance restriction (so prediction will be slower).

To combine these ideas in a manner that can come with a theoretical guarantee, Chen extended an existing approach designed for classification and regression called *kernel netting* [Kpotufe and Verma, 2017] to time-to-event prediction. Chen referred to the resulting class of models as *survival kernets* (the latter word abbreviates "kernel netting").

We now state the training and prediction procedures for survival kernets, which depends on the hyperparameters $\varepsilon \geq 0$ and $\tau > 0$ mentioned above. Recall that we have discretized time to the user-specified grid $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$. Let $\mathcal{D}_1 \subseteq [n]$ and $\mathcal{D}_2 \subseteq [n]$ be subsets of the training data (*e.g.*, we randomly divide the n training data into two halves $\mathcal{D}_1$ and $\mathcal{D}_2$).

Training. We proceed as follows:

1. (Learn kernel function) Train a DKSA model on dataset $\mathcal{D}_1$ by using a neural network optimizer to minimize $\mathbf{L}_{\text{DKSA-NLL}}(\theta)$ (restricted to using only training data in $\mathcal{D}_1$). Denote the resulting estimate of θ as $\widehat{\theta}$.

2. (Compute embedding vectors) Recall that $\mathbf{K}(x, x'; \theta) = \exp(-\|\mathbf{f}(x;\theta) - \mathbf{f}(x';\theta)\|^2)$ (equation (52)). In particular, the distance $\|\mathbf{f}(x;\theta) - \mathbf{f}(x';\theta)\|$ is computed in the latent embedding space. We compute the embedding vectors $\widetilde{X}_i = \mathbf{f}(X_i; \widehat{\theta}) \in \mathbb{R}^{d_{\text{emb}}}$ for each $i \in \mathcal{D}_2$.

3. (Cluster embedding vectors) Run an exemplar-based clustering method on the embedding vectors $\{\widetilde{X}_i : i \in \mathcal{D}_2\}$ to obtain a set of exemplars $\mathcal{Q} \subseteq \mathcal{D}_2$.

   While there are different clustering methods that are possible here, to get a theoretical guarantee, the clustering method used by Chen [2024] is the standard ε-net method (see, for instance, the textbook by Vershynin [2018]), which has a hyperparameter $\varepsilon \geq 0$ (where choosing $\varepsilon = 0$ results in every point being its own cluster, and as $\varepsilon \to \infty$, we will have all points in the same cluster):

   (a) Initialize $\mathcal{Q} := 0$.

   (b) For $i \in \mathcal{D}_2$:

      i. If $\mathcal{Q}$ is empty, then add i to the set of exemplars $\mathcal{Q}$. This means that now point i forms a new cluster.

      ii. Otherwise:

A. Let j be the point in $\mathcal{Q}$ whose embedding vector is closest (in Euclidean distance) to $\widetilde{X}_i$.
B. If $\|\widetilde{X}_i - \widetilde{X}_j\| > \varepsilon$: Add i to the set of exemplars $\mathcal{Q}$, so that point i forms a new cluster.

   Otherwise: assign point i to the cluster of exemplar j.

4. (Compute summary information per cluster) For each exemplar $j \in \mathcal{Q}$, let $\mathcal{C}_j \subseteq \mathcal{D}_2$ denote the points that have been assigned to the same cluster as j. Per exemplar j, we compute two summary functions:

$$D_{\text{cluster}}[\ell|j] := \sum_{i \in \mathcal{C}_j} \mathbb{1}\{Y_i = \tau_{(\ell)}\}\Delta_i,$$
$$N_{\text{cluster}}[\ell|j] := \sum_{i \in \mathcal{C}_j} \mathbb{1}\{Y_i \geq \tau_{(\ell)}\}.$$

   These are just the death and at-risk counts from equations (29) and (30) restricted to points in the cluster of exemplar j.

5. (Optional summary information fine-tuning) As an optional step, Chen [2024, Section 3.3] described a method that fine-tunes the summary functions D_{cluster} and N_{cluster} in a neural network framework. For simplicity, we do not go over this fine-tuning step in this monograph.

Prediction. After model training, for a specific test raw input x, we predict $S(\cdot|x)$ as follows.

1. Compute $\widetilde{x} := \mathbf{f}(x; \widehat{\theta})$.
2. Use a nearest neighbor search algorithm to find all exemplars in $\mathcal{Q}$ whose embedding vectors are within distance τ from $\widetilde{x}$. Denote this resulting set of exemplars as $\mathcal{Q}(x; \tau)$.
3. Compute the weighted number of deaths and the weighted number of points at risk of dying at each time index $\ell \in [L]$ as follows:

$$D_{\text{kernet}}[\ell|x] := \sum_{j \in \mathcal{Q}(x;\tau)} \mathbf{K}(x, X_j; \widehat{\theta}) D_{\text{cluster}}[\ell|j],$$
$$N_{\text{kernet}}[\ell|x] := \sum_{j \in \mathcal{Q}(x;\tau)} \mathbf{K}(x, X_j; \widehat{\theta}) N_{\text{cluster}}[\ell|j].$$

4. Finally, we predict $S(\cdot|x)$ using

$$\widehat{S}_{\text{kernet}}(t|x) := \prod_{m=1}^{L} \left(1 - \frac{D_{\text{kernet}}[\ell|x]}{N_{\text{kernet}}[\ell|x]}\right)^{\mathbb{1}\{\tau_{(m)} \leq t\}} \quad \text{for } t \geq 0.$$

If we set $\mathcal{D}_1 = \mathcal{D}_2 = [n]$ (*i.e.*, $\mathcal{D}_1$ and $\mathcal{D}_2$ are both set to be the full training data), $\varepsilon = 0$, and $\tau = \infty$, and we do not use the optional summary information fine-tuning step that is mentioned, then the resulting survival kernet model would make the same prediction as the original DKSA model (using equation (54)). Meanwhile, if $\varepsilon \to \infty$ (so that there is only a single cluster), $\tau \to \infty$ (so that for any test point, we predict using the only cluster present), and we do not use the optional summary information fine-tuning, then the resulting survival kernet model would just become the classical Kaplan-Meier estimator.

In his experiments, Chen [2024] found that setting $\mathcal{D}_1 = \mathcal{D}_2 = [n]$ and using the optional summary information fine-tuning tends to result in the survival kernet model that achieves the highest time-dependent concordance index in practice.

We provide a Jupyter notebook that implements survival kernets.[22] Note that our notebook also shows how to warm-start neural network training with the help of XGBoost, using the strategy by Chen [2024, Section 4].

Model interpretation. Ignoring the optional summary information fine-tuning step, when a survival kernet model makes a prediction for a test raw input x, the only training data that contribute to the prediction are the ones in $\mathcal{Q}(x;\tau)$. In particular, suppose for a moment that $\mathcal{Q}(x;\tau)$ consists of only a single point $q \in [n]$ (which we could interpret as x being "purely explained" by exemplar q's cluster according to the learned survival kernet model). Then we would have

$$\begin{aligned} D_{\text{kernet}}[\ell|x] &= \mathbf{K}(x, X_q; \widehat{\theta}) D_{\text{cluster}}[\ell|q], \\ N_{\text{kernet}}[\ell|x] &= \mathbf{K}(x, X_q; \widehat{\theta}) N_{\text{cluster}}[\ell|q]. \end{aligned}$$

This means that

$$\begin{aligned} \widehat{S}_{\text{kernet}}(t|x) &= \prod_{m=1}^{L} \Big(1 - \frac{\mathbf{K}(x, X_q; \widehat{\theta}) D_{\text{cluster}}[\ell|q]}{\mathbf{K}(x, X_q; \widehat{\theta}) N_{\text{cluster}}[\ell|q]}\Big)^{\mathbb{1}\{\tau_{(m)} \le t\}} \\ &= \prod_{m=1}^{L} \Big(1 - \frac{D_{\text{cluster}}[\ell|q]}{N_{\text{cluster}}[\ell|q]}\Big)^{\mathbb{1}\{\tau_{(m)} \le t\}}, \end{aligned}$$

which is just the Kaplan-Meier estimator restricted to the points assigned to the cluster of exemplar q.

Thus, since any data point purely explained by a single cluster has a prediction given by the Kaplan-Meier for that cluster, a straightforward way to visualize the different clusters is to just overlay their Kaplan-Meier survival functions over each other (we could make such a visualization for any subset of clusters, such as the five largest clusters found). As an example of this, see Figure 5, where we also display: (i) 95% confidence intervals per Kaplan-Meier survival function estimate (computed using the standard exponential Greenwood formula [Kalbfleisch and Prentice, 1980]), (ii) an estimate of each cluster's median survival time (the time when the survival function crosses probability $1/2$, as discussed in Section 2.2.1), and (iii) each cluster's size.

In the case where that raw input space corresponds to fixed-length feature vectors (*e.g.*, $\mathcal{X} = \mathbb{R}^d$), Chen [2024] also proposed a heat map visualization that shows, for each cluster, what values each feature tends to take on. An example of this is shown in Figure 6, where rows correspond to values that the features can take on (note that continuous features have been discretized for visualization purposes) and the columns correspond to different clusters.

When making a prediction for test raw input x, we could determine which clusters contribute to the prediction for x (which are precisely the clusters corresponding to the exemplars in $\mathcal{Q}(x,\tau)$). For only these clusters, we could make visualizations like the ones in Figures 5 and 6 as well as report the similarity scores $\mathbf{K}(x, X_j; \widehat{\theta})$ for each $j \in \mathcal{Q}(x,\tau)$.

Theory. Under fairly general conditions, Chen [2024] showed that the survival kernet model's predicted survival curve converges to $S(t|x)$ for times t that are not too large. To show this result, Chen's analysis requires the training data subsets $\mathcal{D}_1$ and $\mathcal{D}_2$ to be independent of each other (*e.g.*, two disjoint halves of the full training data) and the optional summary information fine-tuning step cannot be used. Unfortunately, this theory does not help with analyzing the best-performing variant of survival kernets that Chen found in his experimental results, which sets $\mathcal{D}_1 = \mathcal{D}_2 = [n]$ and uses the summary information fine-tuning step.

5 Neural Ordinary Differential Equation Formulation of Time-to-Event Prediction

As we discussed in Section 2, modeling time-to-event outcomes in continuous time can be challenging in that computing likelihoods involves evaluating integrals. In particular, we can write the

[22] https://github.com/georgehc/survival-intro/blob/main/S4.3_Survival_Kernets.ipynb

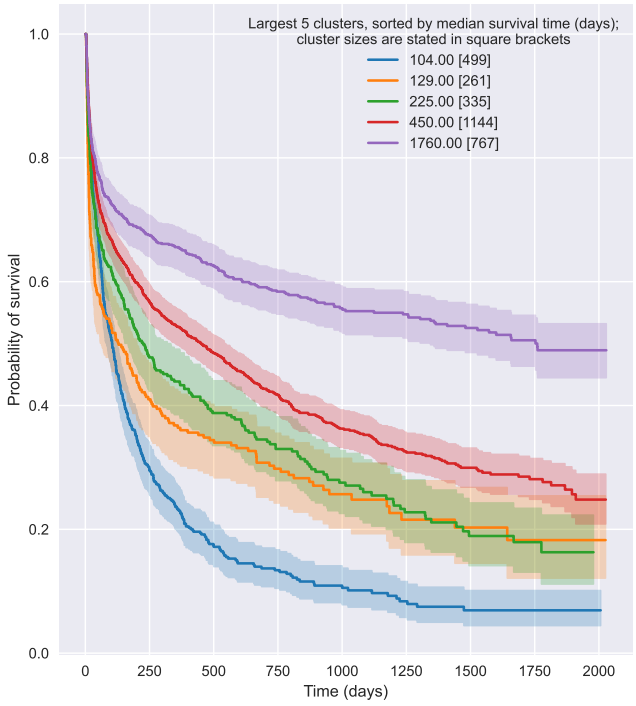


Figure 5: (Figure source: Chen 2024, Figure 1) For the largest 5 clusters found by a survival kernet model on the SUPPORT dataset [Knaus et al., 1995], these are the clusters' Kaplan-Meier survival function plots overlaid over each other.

likelihood in equation (5) as

$$\begin{aligned}\mathcal{L} &= \prod_{i=1}^{n} \{f(Y_i|X_i)^{\Delta_i} S(Y_i|X_i)^{1-\Delta_i}\} \\ &= \prod_{i=1}^{n} \left\{ f(Y_i|X_i)^{\Delta_i} \left[\int_{Y_i}^{\infty} f(u|X_i) \mathrm{d}u \right]^{1-\Delta_i} \right\},\end{aligned}$$

or if we specify the likelihood using the hazard function, then

$$\mathcal{L} = \prod_{i=1}^{n} \left\{ h(Y_i|X_i)^{\Delta_i} \exp\left(- \int_0^{Y_i} h(u|X_i) \mathrm{d}u \right) \right\}. \qquad \text{(9, partially reproduced)}$$

If $f(\cdot|x)$ or $h(\cdot|x)$ can be integrated in closed form, then computing the likelihood functions is straightforward. However, requiring either of these functions to have a closed-form integral could be a restrictive modeling assumption.

A solution that accommodates functions $f(\cdot|x)$ or $h(\cdot|x)$ that lack closed-form integral expressions and that still models time to be continuous is to use neural ordinary differential equations (ODEs) [Chen et al., 2018]. In a nutshell, by using neural ODEs, we could use the continuous time likelihood expression as written (importantly, we leave the integral expression as is without stating what it explicitly evaluates to, as we shall let an ODE solver compute this integral), and it turns out that it is still possible to use minibatch gradient descent for learning model parameters!

As a point of reference, recall that nonparametric methods like conditional Kaplan-Meier estimators or how the baseline hazard function is estimated for the semiparametric Cox model still fundamentally view time as discrete, so that some interpolation is needed to reason about times that are not along the discrete time grid. For example, we had already mentioned that deep kernel survival analysis models in practice depend heavily on how the modeler chooses to discretize time

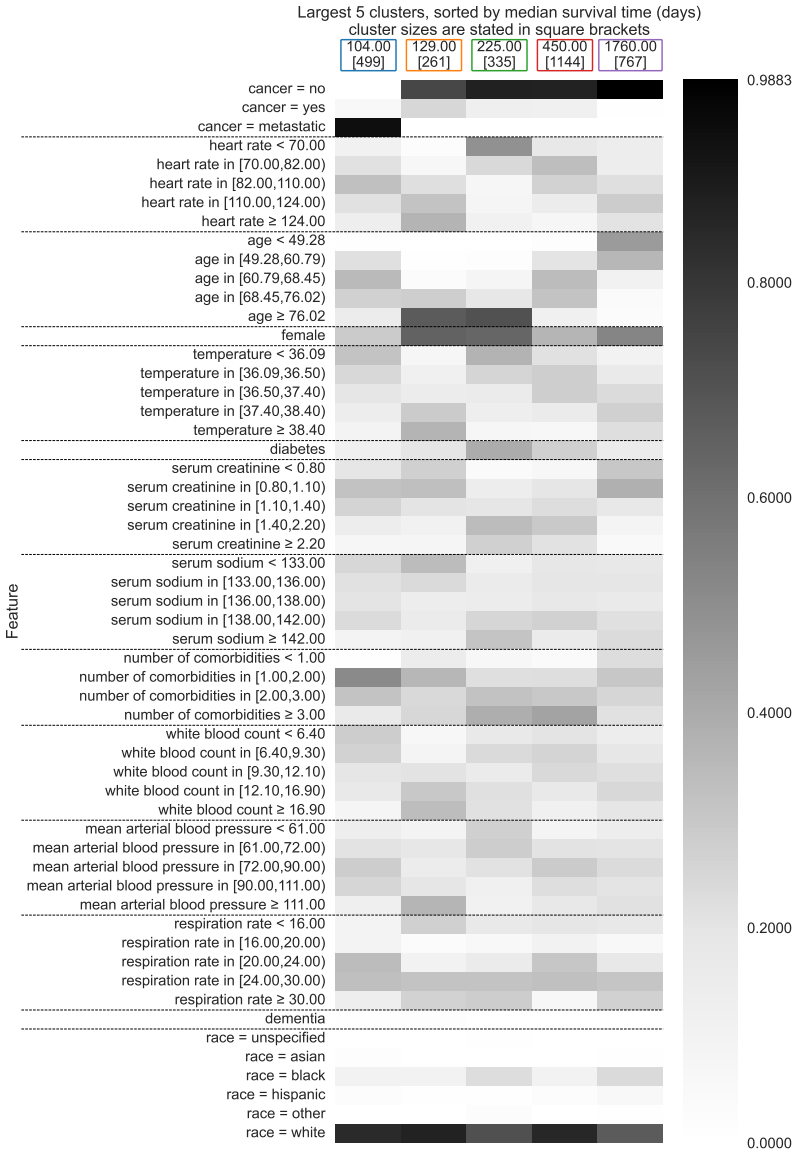


Figure 6: (Figure source: Chen 2024, Figure 1) For the same clusters as the ones in Figure 5, this heatmap shows the prevalence of raw feature values per cluster.

(during training) and to interpolate time (during prediction). Neural ODE time-to-event prediction models move these time discretization and interpolation steps "under the hood" so that the modeler does not have to worry about time discretization issues. Instead, these issues are handled by an ODE solver, which we treat as a black box.

At this point, a number of time-to-event prediction models are available based on neural ODEs (*e.g.*, Groha et al. 2020, Tang et al. 2022b, Danks and Yau 2022, Moon et al. 2022). We cover one example of a neural ODE time-to-event prediction model called SODEN (Survival model through Ordinary Differential Equation Networks), proposed by Tang et al. [2022b]. In their experiments, Tang *et al.* found SODEN to significantly outperform DeepSurv and Cox-Time while being competitive with DeepHit. SODEN is very general and encompasses essentially all the models we have covered thus far including the discrete time models (even though we no longer have to manually discretize time when using neural ODEs, we could still manually discretize time as we show later). We say "essentially all" rather than "all" because as we shall see, the neural ODE versions of many models we have already talked about previously might have some minor differences, and they will be trained differently (with the help of an ODE solver).

As a reminder, in Section 1, we already stated that even though SODEN is very general, we are not always best off using it. As our accompanying code hopefully makes clear, SODEN's training procedure is slow compared to the training procedures we covered in Sections 2 to 4. Also, from playing with the code, one can occasionally encounter numerical stability issues with the ODE solver used.

The rest of this section is organized as follows:

- (Section 5.1) We first go over the ODE formulation Tang *et al.* used and show how it can represent all the models we have discussed so far in this monograph as special cases.
- (Section 5.3) We then explain how training and prediction work with SODEN.
- (Section 5.4) We briefly mention an alternative to using ODEs for handling the integral of equation (9). Specifically, we outline an approach called SurvivalMonotonic-net (SuMo-net) by Rindt et al. [2022].

The main reason we have chosen to focus our exposition on ODEs is that the ODE formulation can readily be related to other models covered earlier in this monograph.

5.1 General ODE Formulation: SODEN

We present a special case of SODEN [Tang et al., 2022b] that is a little easier to describe and corresponds to our problem setup from Section 2. Recall from Summary 2.1 that $\frac{\mathrm{d}}{\mathrm{d}t}H(t|x) = h(t|x)$ and that $H(0|x) = \int_0^0 h(u|x)\mathrm{d}u = 0$. We use these two constraints to define the following ODE for any $x \in \mathcal{X}$:

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}H(t|x) = \mathbf{h}\big((t, H(t|x), x); \theta\big) & \text{for } t > 0, \\ H(0|x) = 0 & \text{(initial condition at time } t = 0), \end{cases} \tag{56}$$

where $\mathbf{h}(\cdot;\theta)$ is a neural network with parameter variable θ. Since $\frac{\mathrm{d}}{\mathrm{d}t}H(t|x) = h(t|x)$, this means that $\mathbf{h}(\cdot;\theta)$ models the hazard function, so its output needs to be a nonnegative number. As our notation indicates, $\mathbf{h}(\cdot;\theta)$ takes three inputs: time t, the cumulative hazard value $H(t|x)$ at time t, and a raw input x. The solution to the ODE is precisely the cumulative hazard function $H(\cdot|x)$. Let's look at a concrete example.

Example 5.1 (Weibull time-to-event prediction model as the solution to an ODE)**.** Let $\mathcal{X} = \mathbb{R}^d$. Let

$$\mathbf{h}\big((t, H(t|x), x); \theta\big) := \big(H(t|x)\big)^{1-e^{-\phi}} e^{\beta^\top x + \phi + \psi e^{-\phi}}, \tag{57}$$

where $\beta \in \mathbb{R}^d$, $\psi \in \mathbb{R}$ and $\phi \in \mathbb{R}$ are parameters (*i.e.*, $\theta = (\beta, \psi, \phi)$). Plugging equation (57)

into equation (56) yields the ODE

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}H(t|x) = \big(H(t|x)\big)^{1-e^{-\phi}} e^{\beta^\top x + \phi + \psi e^{-\phi}} & \text{for } t > 0, \\ H(0|x) = 0 \qquad \text{(initial condition at time } t = 0). \end{cases} \tag{58}$$

This ODE can be solved in closed form (as shown in Section 5.A.1), yielding the solution

$$H(t|x) = e^{(\beta^\top x)e^{\phi} + \psi} t^{e^{\phi}},$$

which is precisely the cumulative hazard function that we had derived for the Weibull time-to-event prediction model (see equation (13)).

As a reminder, we had pointed out after we first presented the Weibull time-to-event prediction model (Example 2.3) that this model is both a proportional hazards model and an accelerated failure time (AFT) model (and moreover, it generalizes the exponential time-to-event prediction model we had presented in Examples 2.1 and 2.2 which correspond to the case where $\phi = 0$). In fact, deep proportional hazards and deep AFT models can both be encoded as special cases of SODEN, as we show shortly.

Note that back in Example 2.3, we already derived a way to learn parameters of the Weibull time-to-event prediction model using maximum likelihood. We will be able to learn these parameters using maximum likelihood under this neural ODE framework as well, where the difference is that the learning procedure relies on an ODE solver.

We now give some examples of families of time-to-event prediction models that are handled by SODEN to illustrate its level of generality.

5.1.1 Special Case: Deep Proportional Hazards Models

To encode a deep proportional hazards model that is fully parametric in SODEN, it suffices to set

$$\mathbf{h}\big((t, H(t|x), x); \theta\big) := \mathbf{h}_0(t; \theta) e^{\mathbf{f}(x;\theta)},$$

where $\mathbf{h}_0(\cdot;\theta) : [0,\infty) \to [0,\infty)$ and $\mathbf{f}(\cdot;\theta) : \mathcal{X} \to \mathbb{R}$ are two different user-specified neural networks with parameter variable θ. Since $\mathbf{h}(\cdot;\theta)$ models the hazard function, by construction, we have chosen it to satisfy the proportional hazards assumption (equation (39)). Once we have specified $\mathbf{h}\big((t, H(t|x), x); \theta\big)$, we can then use SODEN's training procedure that depends on an ODE solver (described momentarily in Section 5.3); in fact, this resulting method is called SODEN-PH in the original SODEN paper [Tang et al., 2022b]. We emphasize that from a modeling perspective, a fully parametric deep proportional hazards model does not actually need to be encoded in an ODE framework for us to learn the model parameters (since we can use the training procedure described in Section 3.2 instead). However, we are simply showing that it is also possible to learn this same model class within the SODEN framework with a different training procedure.

To encode a deep proportional hazards model that is semiparametric, we could still use the above formulation, but we instead set $\mathbf{h}_0(\cdot;\theta)$ to be a piecewise constant function. Let $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$ be the unique times of death, and set $\tau_{(0)} := 0$. We introduce unconstrained parameters $\gamma_1, \gamma_2, \ldots, \gamma_L \in \mathbb{R}$ (so that θ now includes $\gamma_1, \ldots, \gamma_L$). Then we define

$$\mathbf{h}_0(t;\theta) := \begin{cases} g(\gamma_\ell) & \text{if } \tau_{(\ell-1)} < t \le \tau_{(\ell)} \text{ for } \ell \in [L], \\ 0 & \text{if } t > \tau_{(L)}, \end{cases}$$

where $g(\cdot)$ is any activation function that outputs a nonnegative number (*e.g.*, ReLU, softplus); recall that in continuous time, hazard function values could be larger than 1, so we do not need to enforce an upper bound constraint (see Remark 3.3). This semiparametric model could then be learned using SODEN's training procedure (Section 5.3) and, in particular, we do not need to use the two-step procedure from Section 3.3.

Note that even though we motivated the use of neural ODEs as a way to not have to explicitly discretize time, our semiparametric example here emphasizes that we could still intentionally set $\mathbf{h}_0(\cdot;\theta)$ to use a discrete time grid. Separately, we could of course replace $\mathbf{f}(x;\theta)$ with a neural network that depends on both x and t to get a model like Cox-Time (Section 3.4).

5.1.2 Special Case: Deep Accelerated Failure Time Models

As another example of a wide family of models that SODEN encompasses, we look at deep AFT models. In general, these assume that the survival function is of the form

$$S(t|x) = \mathbf{S}_0(te^{\mathbf{f}(x;\theta)};\theta) \quad \text{for } t \geq 0, x \in \mathcal{X}, \tag{59}$$

where $\mathbf{S}_0(\cdot;\theta) : [0,\infty) \to [0,1]$ and $\mathbf{f}(\cdot;\theta) : \mathcal{X} \to \mathbb{R}$ are neural networks with parameter variable θ. Note that $\mathbf{S}_0(\cdot;\theta)$ is a survival function (so that it monotonically decays from 1 to 0). The above property implies that for all $x \in \mathcal{X}$, the survival function $S(\cdot|x)$ has the same shape (namely that of $\mathbf{S}_0(\cdot;\theta)$) except where we stretch the time axis by a factor of $e^{\mathbf{f}(x;\theta)}$. When $\mathbf{f}(x;\theta)$ is larger, we accelerate how fast death is likely to happen. Note that "classical" AFT models [Prentice and Kalbfleisch, 1979] correspond to the case where $\mathcal{X} = \mathbb{R}^d$ and $\mathbf{f}(x;\theta) = \theta^\top x$ for parameter vector $\theta \in \mathbb{R}^d$, and $\mathbf{S}_0(\cdot;\theta)$ could either be parametric or left unspecified.[23]

Example 5.2 (Weibull time-to-event prediction model is an AFT model)**.** The Weibull time-to-event prediction model corresponds to the case where $\mathcal{X} = \mathbb{R}^d$, $\mathbf{S}_0(t;\theta) = \exp(-e^{\psi}t^{e^{\phi}})$, and $\mathbf{f}(x;\theta) = \beta^\top x$, where $\theta = (\beta, \psi, \phi) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}$.

Equation (59) is equivalent to the condition

$$h(t|x) = \mathbf{h}_0(te^{\mathbf{f}(x;\theta)};\theta)e^{\mathbf{f}(x;\theta)} \quad \text{for } t \geq 0, x \in \mathcal{X}, \tag{60}$$

where $\mathbf{h}_0(t;\theta) := -\frac{\mathrm{d}}{\mathrm{d}t}\log \mathbf{S}_0(t;\theta)$ (for a derivation of this equivalence, see Section 5.A.2). Thus, in the SODEN model, we could represent a deep AFT model by setting

$$\mathbf{h}\big((t, H(t|x), x);\theta\big) := \mathbf{h}_0(te^{\mathbf{f}(x;\theta)};\theta)e^{\mathbf{f}(x;\theta)},$$

where the user specifies the neural networks $\mathbf{h}_0(\cdot;\theta)$ and $\mathbf{f}(\cdot;\theta)$.

Tang et al. [2022a] suggested an alternative approach to specifying deep AFT models in an ODE framework (note that they actually suggested it for a classical rather than deep AFT model, but the idea trivially extends to the deep AFT case). In particular, it turns out that if we set

$$\mathbf{h}\big((t, H(t|x), x);\theta\big) := \mathbf{g}(H(t|x);\theta)e^{\mathbf{f}(x;\theta)},$$

for some user-specified neural networks $\mathbf{g}(\cdot;\theta)$ and $\mathbf{f}(\cdot;\theta)$, then the solution to the ODE is a deep AFT model. In fact, we already saw this way of specifying an AFT model in Example 5.1, where $\mathbf{g}(t;\theta) := e^{\phi+\psi e^{-\phi}}t^{1-e^{-\phi}}$, and $\mathbf{f}(x;\theta) = \beta^\top x$. The function $\mathbf{g}(\cdot;\theta)$ relates to the shape of $\mathbf{S}_0(\cdot;\theta)$ in a nontrivial manner, although it is possible to convert between these two functions (for details, see Section 2.2 of Tang et al. [2022a]).

5.1.3 Special Case: Deep Extended Hazard Models

A family of models that contains both deep proportional hazards and deep AFT models as special cases is called *deep extended hazard models* (DeepEH) [Zhong et al., 2021]. As the basic idea is straightforward, we directly state how to set $\mathbf{h}(\cdot;\theta)$ for SODEN to get a DeepEH model:

$$\mathbf{h}\big((t, H(t|x), x);\theta\big) := \mathbf{h}_0(te^{\mathbf{f}_1(x;\theta)};\theta)e^{\mathbf{f}_2(x;\theta)},$$

[23] As we pointed out in footnote 10 on page 21, for more details on classical AFT models, please see Chapter 12 of the textbook by Klein and Moeschberger [2003] or Chapter 3 of the textbook by Box-Steffensmeier and Jones [2004].

where $\mathbf{h}_0(\cdot;\theta)$, $\mathbf{f}_1(\cdot;\theta)$, and $\mathbf{f}_2(\cdot;\theta)$ are neural networks with parameter variable θ. When $\mathbf{f}_1(x;\theta) = 0$, then we get a deep proportional hazards model (see equation (39)). If instead $\mathbf{f}_1(x;\theta) = \mathbf{f}_2(x;\theta)$, then we get a deep AFT model (see equation (60)). Note that the way Zhong et al. [2021] learn a DeepEH model has a known theoretical guarantee, whereas if we learn it using the SODEN learning procedure, there is no known theoretical guarantee that we are aware of.

5.2 Special Case: Converting Discrete Time Models to Continuous Time

As we saw for deep proportional hazards models in Section 5.1.1, we could set the baseline hazard function $\mathbf{h}_0(\cdot;\theta)$ to be piecewise constant over a discrete time grid. We could use this same idea to directly specify the hazard function $\mathbf{h}(\cdot;\theta)$ as piecewise constant over a user-specified grid $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$ (these need not be the unique times of death and could be chosen by other strategies) with $\tau_{(0)} := 0$. In fact, by making this piecewise constant assumption, we could actually convert any time-to-event prediction model specified in discrete time (such as the ones in Section 2.3 as well as deep kernel Kaplan-Meier estimators of Section 4) into a continuous time model.

To give an example of how to convert a discrete time model to continuous time, consider Nnet-survival [Gensheimer and Narasimhan, 2019], which we covered in Example 2.5. For this model, we could set the SODEN parametric hazard function to be

$$\begin{aligned}&\mathbf{h}\big((t, H(t|x), x);\theta\big)\\ &:= \begin{cases} \Big(\frac{1}{\tau_{(\ell)} - \tau_{(\ell-1)}}\Big)\mathbf{h}[\ell|x;\theta] & \text{if } \tau_{(\ell-1)} < t \leq \tau_{(\ell)} \text{ for } \ell \in [L],\\ 0 & \text{if } t > \tau_{(L)}, \end{cases}\end{aligned}$$

where $\mathbf{h}[\cdot|x;\theta]$ is given in equation (23). Note that in this case, the ODE is actually straightforward to solve since $\mathbf{h}\big((t, H(t|x), x);\theta\big)$ does not depend on $H(t|x)$. Then note that by integrating from time 0 to time $\tau_{(\ell)}$ for $\ell \in [L]$, we get

$$H(\tau_{(\ell)}|x) = \int_0^{\tau_{(\ell)}} \mathbf{h}\big((u, H(u|x), x);\theta\big)\mathrm{d}u = \sum_{m=1}^{\ell} \mathbf{h}[m|x;\theta].$$

Perhaps what is more interesting is that the model would now interpolate. Suppose that time $t \in (\tau_{(\ell-1)}, \tau_{(\ell)}]$. Then

$$\begin{aligned}H(t|x) &= \int_0^{\tau_{(\ell)}} \mathbf{h}\big((u, H(u|x), x);\theta\big)\mathrm{d}u\\ &= \sum_{m=1}^{\ell-1} \mathbf{h}[m|x;\theta] + \left(\frac{t - \tau_{(\ell-1)}}{\tau_{(\ell)} - \tau_{(\ell-1)}}\right)\mathbf{h}[\ell|x;\theta].\end{aligned}$$

This is precisely using an interpolation strategy that assumes a piecewise constant hazard function.

In terms of how the prediction targets relate between continuous and discrete time, first note that $H[\ell|x] = H(\tau_{(\ell)}|x)$. However, $h[\ell|x]$ would in general not equal $h(\tau_{(\ell)}|x) = \frac{1}{\tau_{(\ell)} - \tau_{(\ell-1)}}\mathbf{h}[\ell|x;\theta]$ due to the extra multiplicative factor in the latter; instead,

$$h[\ell|x] = \mathbf{h}[\ell|x;\theta] = h(\tau_{(\ell)}|x) \cdot (\tau_{(\ell)} - \tau_{(\ell-1)}) \quad \text{for } \ell \in [L].$$

Meanwhile, $S[\ell|x;\theta]$ would also in general not equal $S(\tau_{(\ell)}|x) = \exp(-H(\tau_{(\ell)}|x)) = \exp(-H[\ell|x])$ due to Proposition 2.1. However, this proposition shows the precise manner in which $S(\tau_{(\ell)}|x) = \exp(-H[\ell|x])$ approximates $S[\ell|x]$.

Since deep kernel survival analysis could be viewed as parameterizing a discrete time hazard function (equation (53)), converting a deep kernel survival analysis model to continuous time

would work the same way as what we just showed for Nnet-survival. The main difference is that we would also have to remember to convert the leave-one-out discrete time hazard functions (equation (55)) during model training.

The same conversion strategy could be used with the simplified version of DeepHit [Lee et al., 2018] that we presented in Example 2.4, with just a minor difference: DeepHit parameterizes the distribution $\mathbb{P}_{T|X}(\cdot|x)$ in terms of the survival time PMF $f[\cdot|x]$ and not the hazard function $h[\cdot|x]$. However, we could simply use Summary 2.2 to obtain

$$h[\ell|x] = \frac{f[\ell|x]}{S[\ell-1|x]} = \frac{f[\ell|x]}{\sum_{m=\ell}^{L} f[m|x]},$$

which means that by having a parametric form of $f[\cdot|x]$, we have a parametric form for $h[\cdot|x]$.

Overall, converting discrete time models to continuous time models using a neural ODE framework is possible and gives a different way of learning such discrete time models (by using the general learning procedure for SODEN) that automatically also handles interpolation. However, in such cases, solving the maximum likelihood problem directly in discrete time could be faster in practice as there is no need to use an ODE solver. After training a discrete time model, we could also use piecewise constant hazard interpolation to back out continuous time predictions.

5.3 Prediction and Training with a SODEN Model

We now give an overview of how to train a SODEN model and how to subsequently make predictions.

Training. Training neural ODEs (such as the one in equation (56)) is possible thanks to the landmark paper by Chen et al. [2018]. Importantly, using any user-specified ODE solver, given any raw input $x \in \mathcal{X}$ and neural network parameters θ, we can numerically solve the ODE in equation (56) (going from time 0 to any user-specified time $t > 0$) to obtain an estimate for $H(t|x)$. We denote the resulting estimate as $H_{\text{ODE-solve}}(t|x;\theta)$. Then a major result of Chen et al. [2018] is that the loss function we use can contain the terms $\mathbf{h}((t, H(t|x), x);\theta)$ and $H_{\text{ODE-solve}}(t|x;\theta)$, where it is possible to compute the gradient of $H_{\text{ODE-solve}}(t|x;\theta)$ with respect to θ (Chen *et al.* provide the software package `torchdiffeq` for computing such gradients).

Then to train the SODEN model, Tang et al. [2022b] simply use the negative log likelihood loss we had stated in equation (11) except we replace $\mathbf{h}(t|X_i;\theta)$ with $\mathbf{h}((Y_i, H(Y_i|X_i), X_i);\theta)$ and we replace the integral (which is equal to $H(Y_i|X_i)$) with $H_{\text{ODE-solve}}(Y_i|X_i;\theta)$. The resulting loss is

$$\begin{aligned}&\mathbf{L}_{\text{SODEN-NLL}}(\theta)\\ &:= -\frac{1}{n}\sum_{i=1}^{n}\{\Delta_i \log \mathbf{h}\big((Y_i, H(Y_i|X_i), X_i);\theta\big) - H_{\text{ODE-solve}}(Y_i|X_i;\theta)\}.\end{aligned}$$

We can use a neural network optimizer to minimize the loss to obtain an estimate $\widehat{\theta} := \arg\min_\theta \mathbf{L}_{\text{SODEN-NLL}}(\theta)$.

Prediction. For any test raw input $x \in \mathcal{X}$, by using the user-specified ODE solver, we can predict the cumulative hazard function using $\widehat{H}(t|x) := H_{\text{ODE-solve}}(t|x;\widehat{\theta})$. We can then predict the survival function with $\widehat{S}(t|x) := \exp(-\widehat{H}(t|x))$ and the hazard function with $\widehat{h}(t|x) := \mathbf{h}\big((t, \widehat{H}(t|x), x);\widehat{\theta}\big)$.

Note that to predict the hazard function, we first predict the cumulative hazard function since even though neural network $\mathbf{h}(\cdot;\widehat{\theta})$ models the hazard function, recall that it takes in three inputs and in general can depend on the cumulative hazard value at any given time.

We provide a Jupyter notebook that shows how to train a SODEN model and subsequently make predictions with it.[24] Our notebook makes it clear where calls to the ODE solver happen.

[24] `https://github.com/georgehc/survival-intro/blob/main/S5.2_SODEN.ipynb`

5.4 An Alternative to ODEs via Monotonic Networks: SuMo-net

We began Section 5 by mentioning that the right-censored likelihood from Section 2 has the form

$$\mathcal{L} = \prod_{i=1}^{n} \left\{ h(Y_i|X_i)^{\Delta_i} \exp\Big(- \int_0^{Y_i} h(u|X_i)\mathrm{d}u \Big) \right\}, \qquad \text{(9, partially reproduced)}$$

where the difficulty of working with this likelihood is in evaluating the integral. At a high level, the basic idea of SODEN was that we modeled the hazard function h with a neural network $\mathbf{h}(\cdot;\theta)$, and we relied on the neural ODE framework to take care of automatically integrating $h(\cdot|X_i)$ (specified in terms of $\mathbf{h}(\cdot;\theta)$) to compute $H(Y_i|X_i) = \int_0^{Y_i} h(u|X_i)\mathrm{d}u$.

Naturally, this suggests that we could have attempted an alternative parameterization. We can rewrite the likelihood of equation (9) as

$$\mathcal{L} = \prod_{i=1}^{n} \left\{ \Big[\frac{\mathrm{d}H}{\mathrm{d}t}(Y_i|X_i) \Big]^{\Delta_i} \exp\big(- H(Y_i|X_i) \big) \right\}.$$

Then we could directly model the cumulative hazard function $H(\cdot|x)$ with a neural net $\mathbf{H}(\cdot|x;\theta)$ and instead rely on automatic differentiation software to compute $\frac{dH}{dt}(Y_i|X_i) = h(Y_i|X_i)$ (so that now we leave the derivative unspecified in the loss function, whereas previously in the ODE setup, we left the cumulative hazard function unspecified in the loss). However, we need to add the constraint that $\mathbf{H}(t|x;\theta)$ is nonnegative and monotonically increases with respect to time t. We can take advantage of the fact that there already exist standard neural network architectures for enforcing monotonicity (*e.g.*, Chilinski and Silva 2020, Yanagisawa et al. 2022). This approach does not use neural ODEs and simply takes advantage of automatic differentiation, which is already a standard component of all modern neural network software packages. This resulting approach corresponds to the SurvivalMonotonic-Network (SuMo-net) model by Rindt et al. [2022].[25]

Rindt et al. [2022] show that SuMo-net works very well in practice, outperforming neural ODEs (namely, SODEN and also the SurvNODE model by Groha et al. [2020]) on several benchmark datasets in terms of log likelihood scores and also computation time. In terms of computation time, roughly, automatic differentiation is relatively fast (and natively supported by neural network software packages) compared to making many calls to an ODE solver.

The reason that it is straightforward relating SODEN to models we have presented earlier in this monograph is that these models can be specified in terms of a hazard function (continuous or discrete), *i.e.*, there is some way to directly parameterize the hazard function. SuMo-net, on the other hand, could be thought of as asking the modeler to parameterize either the survival or cumulative hazard function directly, but not the hazard function, so that the hazard function is indirectly obtained. Whether this is desirable depends on the use case. Of course, if one just cares about a survival model doing well on a specific evaluation metric and nothing else, then we could just choose whichever model does best on the evaluation metric of interest. However, if there are other design considerations (such as some notion of interpretability), or if the model is not being used purely for prediction but for causal inference, then choosing which model is "best" can be more challenging. We discuss some of these issues in Section 7.

5.A Technical Details

5.A.1 Solving the Weibull Time-to-Event Prediction Model's ODE From Example 5.1

Treating x as fixed, the ODE of equation (58) can be written as

$$\frac{\mathrm{d}}{\mathrm{d}t} y(t) = a \cdot b \cdot y(t)^{1-\frac{1}{a}}$$

[25] Technically, Rindt et al. [2022] specify SuMo-net in terms of constraining the survival function $S(t|x)$ to monotonically decrease in t, but they also explain how to instead directly model the cumulative hazard function $H(t|x)$ (and constrain this to be monotonic), which corresponds to our exposition.

subject to the constraint that $y(0) = 0$ (where in our case, $y(t) = H(t|x)$, $a = e^{\phi}$, and $b = e^{\beta^\top x + \psi e^{-\phi}}$). Rearranging terms, we have

$$y(t)^{\frac{1}{a}-1}\frac{\mathrm{d}}{\mathrm{d}t}y(t) = a \cdot b.$$

Integrate both sides with respect to t to get

$$a(y(t))^{1/a} = a \cdot b \cdot t + c,$$

where c is a constant to be determined based on the constraint $y(0) = 0$. We rearrange terms to get that

$$y(t) = \left(b \cdot t + \frac{c}{a}\right)^a,$$

where, using the constraint, we get that we must have $c = 0$. Hence, $y(t) = (b \cdot t)^a$, *i.e.*,

$$H(t|x) = (e^{\beta^\top x + \psi e^{-\phi}} t)^{e^{\phi}} = e^{(\beta^\top x)e^{\phi} + \psi} t^{e^{\phi}}.$$

5.A.2 Deriving the Hazard Function of Deep AFT Models

We show that the AFT model as defined in equation (59) (namely that $S(t|x) = \mathbf{S}_0(te^{\mathbf{f}(x;\theta)};\theta)$) implies that the hazard function $h(\cdot|x)$ is the one in equation (60), which we reproduce here for convenience:

$$h(t|x) = \mathbf{h}_0(te^{\mathbf{f}(x;\theta)};\theta)e^{\mathbf{f}(x;\theta)} \quad \text{for } t \geq 0, x \in \mathcal{X}. \tag{60, reproduced}$$

To prove that this factorization holds for the hazard function, we begin by stating yet another equivalent characterization of a deep AFT model that will be helpful.

Proposition 5.1 (Log survival time viewpoint of a deep AFT model)**.** Using the time-to-event prediction setup in Section 2.1 and the key assumptions of 2.2, suppose that the random survival time T satisfies the equality

$$\log T = -\mathbf{f}(X;\theta) + W, \tag{61}$$

where the "noise" random variable W is independent of everything else, and e^W has a CDF given by $\mathbf{F}_0(t;\theta) := 1 - \mathbf{S}_0(t;\theta)$ for $t \geq 0$. Then this setup is equivalent to making the assumption that the survival function $S(\cdot|x)$ satisfies the factorization in equation (59), *i.e.*, this time-to-event prediction model is a deep AFT model.

Proof of Proposition 5.1. To see why equation (61) is equivalent to equation (59), first note that equation (61) can be rearranged as $T = e^{-\mathbf{f}(X;\theta)}e^W$. Then,

$$\begin{aligned}
S(t|x) &= \mathbb{P}(T > t|X = x) \\
&= \mathbb{P}(e^{-\mathbf{f}(X;\theta)}e^W > t|X = x) \\
&= \mathbb{P}(e^W > te^{\mathbf{f}(X;\theta)}) \\
&= 1 - \mathbf{F}_0(te^{\mathbf{f}(X;\theta)};\theta) \\
&= \mathbf{S}_0(te^{\mathbf{f}(X;\theta)};\theta),
\end{aligned}$$

which shows that equation (59) holds. We could reverse the steps to show that equation (59) implies equation (61). □

We now proceed to derive the hazard function of a deep AFT model. Summary 2.1 tells us that $h(t|x) = \frac{f(t|x)}{S(t|x)}$ which combined with equation (59) yields

$$h(t|x) = \frac{f(t|x)}{\mathbf{S}_0(te^{\mathbf{f}(x;\theta)};\theta)}. \tag{62}$$

As a reminder, $f(\cdot|x)$ is the PDF of $\mathbb{P}_{T|X}(\cdot|x)$, and the corresponding CDF is $F(t|x) = \int_0^t f(u|x)\mathrm{d}u$. We next write $f(\cdot|x)$ in terms of $\mathbf{f}_0(\cdot;\theta)$, the PDF corresponding to CDF $\mathbf{F}_0(\cdot;\theta)$. To do this, we start by writing CDF $F(\cdot|x)$ in terms of $\mathbf{F}_0(\cdot;\theta)$:

$$\begin{aligned}
F(t|x) &= \mathbb{P}(T \leq t|X = x) \\
&= \mathbb{P}(e^{-\mathbf{f}(X;\theta)+W} \leq t|X = x) \qquad \text{(using equation (61))} \\
&= \mathbb{P}(e^W \leq te^{\mathbf{f}(X;\theta)}|X = x) \\
&= \mathbb{P}(e^W \leq te^{\mathbf{f}(x;\theta)}) \\
&= \mathbf{F}_0(te^{\mathbf{f}(x;\theta)};\theta).
\end{aligned}$$

Then using the derivative chain rule,

$$f(t|x) = \frac{\mathrm{d}F(t|x)}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{F}_0(te^{\mathbf{f}(x;\theta)};\theta) = \mathbf{f}_0(te^{f(x;\theta)};\theta)e^{\mathbf{f}(x;\theta)}. \tag{63}$$

Combining equations (62) and (63), we have

$$h(t|x) = \frac{\mathbf{f}_0(te^{\mathbf{f}(x;\theta)};\theta)e^{\mathbf{f}(x;\theta)}}{\mathbf{S}_0(te^{\mathbf{f}(x;\theta)};\theta)} = \mathbf{h}_0(te^{\mathbf{f}(x;\theta)};\theta)e^{\beta^\top x},$$

where the last step uses the fact that $\mathbf{h}_0(t;\theta) = \frac{\mathbf{f}_0(t;\theta)}{\mathbf{S}_0(t;\theta)}$ for the same reason $h(t|x) = \frac{f(t|x)}{S(t|x)}$ in Summary 2.1. This completes the proof. □

6 Beyond the Basic Time-to-Event Prediction Setup: Multiple Critical Events and Time Series as Raw Inputs

In this section, we present two extensions of the basic time-to-event prediction problem setup we described in Section 2 that showcase concrete directions where deep learning models have been successful. Specifically, we go over the following two extensions that progressively get more general than the standard time-to-event prediction setup:

- (Section 6.1) In previous sections, we focused on modeling the time until a specific critical event (*e.g.*, death) happens. We now consider a more general setup where there are k different critical events that we keep track of. For any data point, we want to reason about the time until the earliest of these events happens, as well as which of the k events it is. This is referred to as the *competing risks* setup since we could think of the k events as competing to see which one happens first. A number of deep learning models have been developed for this setup. We go over one called DeepHit [Lee et al., 2018]. Note that the $k = 1$ case recovers the standard setup from Section 2.
- (Section 6.2) We then turn to the problem of what happens when each data point is actually a time series. As we see more of a time series over time, we could keep making predictions of the time until the earliest of k critical events happens and which event it is. Each training point is a time series, and different training points could be time series of different lengths. We go over an example model that handles this setup called Dynamic-DeepHit [Lee et al., 2019]. In the special case where every time series has length 1 (meaning that per data point, we only see the raw input at a single time point before we aim to make a prediction), we recover the setup of Section 6.1.

6.1 Time-to-Event Prediction with Multiple Critical Events: The Competing Risks Setup

Similar to our exposition in Section 2, we first state the statistical framework (Section 6.1.1) and the prediction problem (Section 6.1.2). These lead to a likelihood expression that we could write

(Section 6.1.3). We then give an example model that maximizes the likelihood (Section 6.1.4), which is the full version of the DeepHit model we encountered in Example 2.4. Because this problem setup is different from earlier sections, how we evaluate model accuracy also is a bit different (Section 6.1.5).

Note that DeepHit is at this point a standard baseline to try in time-to-event prediction problems with competing risks (and even in the standard setup without competing risks). More recently, other models that support competing risks have also been developed (*e.g.*, Nagpal et al. 2021a, Danks and Yau 2022, Jeanselme et al. 2023). Similar to what we have seen with the standard time-to-event prediction setup, for this competing risks setup, no single model is best at this point across all datasets. We present DeepHit because it is the original deep competing risks model developed and is fairly straightforward to explain.

There are classical baselines as well although we only mention them now without explaining how they work (as understanding them is not needed for our monograph). Recall that in the standard setup of Section 2, the Kaplan-Meier estimator would give the same population-level predicted survival function regardless of which test point we look at. The analogue in the competing risks setting is called the Aalen-Johansen estimator [Aalen and Johansen, 1978]. Meanwhile, the Fine-Gray subdistribution hazard model [Fine and Gray, 1999] could be thought as the Cox model analogue in the competing risks setting.

6.1.1 Statistical Framework

We keep track of k different critical events. We still assume that we have n training points $(X_1, Y_1, \Delta_1), \ldots, (X_n, Y_n, \Delta_n)$ like in the standard setup. However, the major difference now is that the event indicator $\Delta_i \in \{0, 1, \ldots, k\}$ takes on more possible values. When $\Delta_i = 0$, then Y_i is the censoring time, just as before. However, if $\Delta_i > 0$, then Δ_i tells us which of the k events happened earliest, and Y_i is equal to the time until this earliest critical event happened.

As before, X denotes the random variable for a generic raw input (with distribution $\mathbb{P}_X$), and C denotes the random variable for the true (possibly unobserved) censoring time corresponding to X (with distribution $\mathbb{P}_{C|X}(\cdot|x)$). However, now the random variable T is a *random vector* taking on values in $[0, \infty)^k$ (with distribution $\mathbb{P}_{T|X}(\cdot|x)$). In particular, $T = (T_1, T_2, \ldots, T_k)$ consists of the times until each of the k different critical events happen.

We assume each (X_i, Y_i, Δ_i) for $i \in [n]$ to be generated i.i.d. as follows:

1. Sample raw input X_i from $\mathbb{P}_X$.
2. Sample vector $T_i = (T_{i,1}, T_{i,2}, \ldots, T_{i,k})$ (true times until the k critical events happen) from $\mathbb{P}_{T|X}(\cdot|X_i)$.
3. Sample true censoring time C_i from $\mathbb{P}_{C|X}(\cdot|X_i)$.
4. Set $Y_i = \min\{T_{i,1}, T_{i,2}, \ldots, T_{i,k}, C_i\}$.
   If $Y_i = C_i$: set $\Delta_i = 0$. Otherwise: set $\Delta_i = \arg\min_{\delta \in [k]} T_{i,\delta}$ (if there is a tie for the smallest time, then break the tie arbitrarily).

This generative procedure allows for the times until the critical events happen to potentially depend on each other. However, conditioned on X_i, we still assume that T_i is independent of C_i just as in the standard setup (which we see since steps 2 and 3 do not depend on each other). Note that the k critical events are "exhaustive" in the sense that they are the only options that could happen (unless none of them happen yet due to censoring), which is implied by step 4.

6.1.2 Prediction Task

For all $x \in \mathcal{X}$, we assume that $\mathbb{P}_{T|X}(\cdot|x)$ exists. Consider the random variable $T_{\text{test}} := (T_{\text{test},1}, T_{\text{test},2}, \ldots, T_{\text{test},k}) \in [0, \infty)^k$ that is sampled from $\mathbb{P}_{T|X}(\cdot|x)$. Define the random variables

$$Y_{\text{test}} := \min\{T_{\text{test},1}, T_{\text{test},2}, \ldots, T_{\text{test},k}\}, \quad \Delta_{\text{test}} := \arg\min_{\delta \in [k]} T_{\text{test},\delta}.$$

Notice that these are defined without sampling a censoring time (whereas we assumed censoring times to be generated for training data).

A common prediction target is the so-called *cumulative incidence function* (CIF) [Gray, 1988, Fine and Gray, 1999] of each event $\delta \in [k]$, which is the probability of event δ happening by time t (where $t \geq 0$):

$$F_\delta(t|x) := \mathbb{P}(Y_{\text{test}} \leq t, \Delta_{\text{test}} = \delta \mid X = x). \tag{64}$$

When the number of critical events is $k = 1$, then there would only be a single CIF to estimate corresponding to the single critical event, and it would actually just correspond to the CDF $F(\cdot|x)$ of the survival time distribution $\mathbb{P}_{T|X}(\cdot|x)$ in our problem setup from Section 2.

If we discretize time using the time grid $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$, then the CIF could be written as

$$F_\delta[\ell|x] := \mathbb{P}(Y_{\text{test}} \leq \tau_{(\ell)}, \Delta_{\text{test}} = \delta \mid X = x) \quad \text{for } \delta \in [k], \ell \in [L], x \in \mathcal{X},$$

from which we can write its PMF version

$$f_\delta[\ell|x] := \mathbb{P}(Y_{\text{test}} = \tau_{(\ell)}, \Delta_{\text{test}} = \delta \mid X = x) \quad \text{for } \delta \in [k], \ell \in [L], x \in \mathcal{X}. \tag{65}$$

By how a CDF and PMF relate, we have $F_\delta[\ell|x] = \sum_{m=1}^{\ell} f_\delta[m|x]$. Meanwhile, since a PMF sums to 1, we have $\sum_{\delta=1}^{k} \sum_{\ell=1}^{L} f_\delta[\ell|x] = 1$.

6.1.3 Likelihood

For simplicity, we only present the discrete time likelihood that does not depend on the censoring distribution:

$$\mathcal{L} := \prod_{i=1}^{n} \left\{ (f_{\Delta_i}[\kappa(Y_i)|X_i])^{\mathbb{1}\{\Delta_i \neq 0\}} \left(1 - \sum_{\delta=1}^{k} F_\delta[\kappa(Y_i)|X_i]\right)^{\mathbb{1}\{\Delta_i = 0\}} \right\}, \tag{66}$$

where, as a reminder, $\kappa(Y_i) \in [L]$ denotes the time index that Y_i corresponds to. To make sense of the likelihood, note that for the i-th point, if it is not censored, then the contribution to the likelihood is the factor $f_{\Delta_i}[\kappa(Y_i)|X_i]$, which is the probability of event Δ_i happening at time index $\kappa(Y_i)$ for raw input X_i. Otherwise, if the i-th point is censored, the contribution to the likelihood is

$$\begin{aligned} 1 - \sum_{\delta=1}^{k} F_\delta[\kappa(Y_i)|X_i] &= 1 - \sum_{\delta=1}^{k} \mathbb{P}(Y_{\text{test}} \leq \kappa(Y_i), \Delta_{\text{test}} = \delta \mid X = X_i) \\ &= 1 - \mathbb{P}(Y_{\text{test}} \leq \kappa(Y_i) \mid X = X_i) \\ &= \mathbb{P}(Y_{\text{test}} > \kappa(Y_i) \mid X = X_i), \end{aligned}$$

which is the probability that the earliest critical event happens after time index $\kappa(Y_i)$ for raw input X_i.

6.1.4 Example Model: the Full Version of DeepHit

We had previously covered a special case of the DeepHit model [Lee et al., 2018] for when the number of critical events is $k = 1$ (Example 2.4). We now present the general case that supports multiple critical events. We parameterize the PMF function (equation (65)) in terms of a neural network $\mathbf{f}(\cdot;\theta) : \mathcal{X} \to [0,1]^{L \times k}$ as follows:

$$\begin{aligned} \begin{bmatrix} f_1[1|x] & f_2[1|x] & \cdots & f_k[1|x] \\ f_1[2|x] & f_2[2|x] & \cdots & f_k[2|x] \\ \vdots & \vdots & \ddots & \vdots \\ f_1[L|x] & f_2[L|x] & \cdots & f_k[L|x] \end{bmatrix} &= \begin{bmatrix} \mathbf{f}_{1,1}(x;\theta) & \mathbf{f}_{2,1}(x;\theta) & \cdots & \mathbf{f}_{k,1}(x;\theta) \\ \mathbf{f}_{1,2}(x;\theta) & \mathbf{f}_{2,2}(x;\theta) & \cdots & \mathbf{f}_{k,2}(x;\theta) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{f}_{1,L}(x;\theta) & \mathbf{f}_{2,L}(x;\theta) & \cdots & \mathbf{f}_{k,L}(x;\theta) \end{bmatrix} \\ &=: \mathbf{f}(x;\theta). \end{aligned} \tag{67}$$

Having the output be a 2D table is not required (we can easily flatten the table and it would contain the same information); we have written it this way for clarity of exposition. Recalling that

$\sum_{\delta=1}^{k}\sum_{\ell=1}^{L} f_\delta[\ell|x] = 1$, we require that the neural network's output always sums to 1, which we can easily get by, for instance, having the neural network output a total of $L \cdot k$ numbers that go through a softmax activation.

By plugging equation (67) into equation (66), we obtain

$$\mathcal{L}(\theta) := \prod_{i=1}^{n}\left\{(\mathbf{f}_{\Delta_i,\kappa(Y_i)}(X_i;\theta))^{\mathbb{1}\{\Delta_i\neq 0\}}\left(1-\sum_{\delta=1}^{k}\underbrace{\sum_{m=1}^{\kappa(Y_i)}\mathbf{f}_{\delta,m}(X_i;\theta)}_{F_\delta[\kappa(Y_i)|X_i]}\right)^{\mathbb{1}\{\Delta_i=0\}}\right\}.$$

Then the negative log likelihood loss averaged across training data is:

$$\begin{aligned}
&\mathbf{L}_{\text{DeepHit-NLL}}(\theta)\\
&\quad := -\frac{1}{n}\log\mathcal{L}(\theta)\\
&\quad := -\frac{1}{n}\sum_{i=1}^{n}\Bigg\{\mathbb{1}\{\Delta_i\neq 0\}\log(\mathbf{f}_{\Delta_i,\kappa(Y_i)}(X_i;\theta))\\
&\qquad\qquad + \mathbb{1}\{\Delta_i = 0\}\log\left(1-\sum_{\delta=1}^{k}\sum_{m=1}^{\kappa(Y_i)}\mathbf{f}_{\delta,m}(X_i;\theta)\right)\Bigg\}.
\end{aligned}$$

Since ranking-based accuracy metrics (Section 2.5.1) are popular, Lee et al. [2018] further introduced a ranking loss term that is motivated by the C^{td} index (Definition 2.2). For event $\delta \in [k]$, we define the set of comparable pairs specific to event δ as

$$\mathcal{E}_\delta := \{(i,j) \in [n]\times[n] : \Delta_i = \delta, Y_i < Y_j\}. \tag{68}$$

In particular, this means that if $(i,j) \in \mathcal{E}_\delta$, then training point i experienced critical event δ as the earliest critical event, and training point j has not experienced any critical event yet. This means that at the earlier time Y_i (which corresponds to time index $\kappa(Y_i)$), the model should predict $F_\delta[\kappa(Y_i)|X_i]$ to be *higher* than $F_\delta[\kappa(Y_i)|X_j]$, *i.e.*, we want $F_\delta[\kappa(Y_i)|X_i] - F_\delta[\kappa(Y_i)|X_j]$ to be large. Note that

$$F_\delta[\kappa(Y_i)|X_i] - F_\delta[\kappa(Y_i)|X_j] = \sum_{m=1}^{\kappa(Y_i)}(\mathbf{f}_{\delta,m}(X_i;\theta) - \mathbf{f}_{\delta,m}(X_j;\theta)). \tag{69}$$

Thus, we want this difference to be large for all $(i,j) \in \mathcal{E}_\delta$, for all $\delta \in [k]$.

With the above intuition, for hyperparameters $\eta = (\eta_1, \eta_2, \ldots, \eta_k) \in [0,\infty)^k$ and $\sigma > 0$, we define the ranking loss term

$$\begin{aligned}
&\mathbf{L}_{\text{DeepHit-ranking}}(\theta;\eta,\sigma)\\
&\quad := \frac{1}{k}\sum_{\delta=1}^{k}\frac{\eta_\delta}{|\mathcal{E}_\delta|}\sum_{(i,i')\in\mathcal{E}_\delta}\exp\left(\frac{\sum_{m=1}^{\kappa(Y_i)}[\mathbf{f}_{\delta,m}(X_{i'};\theta) - \mathbf{f}_{\delta,m}(X_i;\theta)]}{\sigma}\right).
\end{aligned}$$

Having each of the terms being summed as small as possible aims to maximize equation (69). Hyperparameter $\eta_\delta \geq 0$ controls how much we care about ranking for critical event δ, and hyperparameter $\sigma > 0$ controls how much we care about ranking across all critical events (as $\sigma \to \infty$, we stop caring about ranking).

Then the full DeepHit loss is:

$$\mathbf{L}_{\text{DeepHit}}(\theta) := \mathbf{L}_{\text{DeepHit-NLL}}(\theta) + \mathbf{L}_{\text{DeepHit-ranking}}(\theta;\eta,\sigma). \tag{70}$$

Note that our presentation of DeepHit is slightly different from the original version by Lee et al. [2018] in that for each loss term we use, we have included some normalization constants (the fraction $\frac{1}{n}$ in $\mathbf{L}_{\text{DeepHit-NLL}}$, and the fractions $\frac{1}{k}$ and $\frac{1}{|\mathcal{E}_\delta|}$ in $\mathbf{L}_{\text{DeepHit-ranking}}$). Lee *et al.* also choose a specific base neural network architecture, whereas we intentionally leave it up to the user to specify.

We then use a neural network optimizer to solve $\widehat{\theta} := \arg\min_\theta \mathbf{L}_{\text{DeepHit}}(\theta)$. For any test raw input $x \in \mathcal{X}$, we could then predict the PMF form of the CIFs using $\mathbf{f}(x; \widehat{\theta})$ (see equation (67)), from which we could readily recover an estimate for all the critical events' CIFs.

In our companion code repository, we provide a Jupyter notebook that implements the full DeepHit model with competing risks.[26] This notebook builds on the earlier DeepHit Jupyter notebook that we provided a link to in Example 2.4, which was for the standard right-censored survival analysis setup. In the competing risks version, instead of using the SUPPORT dataset [Knaus et al., 1995], we now use the PBC dataset [Fleming and Harrington, 1991], which is on predicting times until death or transplantation of various patients with primary biliary cirrhosis of the liver. Note that the PBC dataset is actually a time series dataset. However, per data point, we only consider the initial time step, so that we reduce it to a tabular dataset.

6.1.5 Evaluation Metrics

We point out a few evaluation metrics that are possible. Note that we state these using the continuous time version of CIFs, which could be converted into discrete time easily.

C^{td} index. First, the C^{td} index (Definition 2.2) generalizes to the competing risks setting by using the set of comparable pairs $\mathcal{E}_\delta$ (equation (68)) so that we now have a C^{td} index score per critical event $\delta \in [k]$.

Definition 6.1 (C^{td} index for competing risks)**.** Let $\delta \in [k]$. Suppose that we have a CIF estimate $\widehat{F}_\delta(\cdot|x)$ for any $x \in \mathcal{X}$. Then using the set of comparable pairs $\mathcal{E}_\delta$ from equation (68), we define the C^{td} index for event δ as

$$C^{\text{td}}_\delta := \frac{1}{|\mathcal{E}_\circ|} \sum_{(i,j)\in\mathcal{E}_\circ} \mathbb{1}\{\widehat{F}_\delta(Y_i|X_i) > \widehat{F}_\delta(Y_i|X_j)\},$$

which is between 0 and 1. Higher scores are better.

Truncated time-dependent concordance index. We can generalize the truncated time-dependent concordance index C^{td}_t (Definition 2.3) to the competing risks setting in the same manner as for the C^{td} index. We define the set of comparable pairs specific to event $\delta \in [k]$ and time $t \geq 0$ as

$$\mathcal{E}_\delta(t) := \{(i,j) \in [n] \times [n] : \Delta_i = \delta, Y_i < t, Y_j > Y_i\}.$$

We then define the following accuracy score.

Definition 6.2 (Truncated time-dependent concordance index for competing risks)**.** Let $\delta \in [k]$ and $t \geq 0$. Suppose that we have a CIF estimate $\widehat{F}_\delta(\cdot|x)$ for any $x \in \mathcal{X}$. Then using the set of comparable pairs $\mathcal{E}_\delta(t)$, we define the truncated time-dependent concordance index for event δ at time t as

$$C^{\text{td}}_{\delta,t} := \frac{\sum_{(i,j)\in\mathcal{E}_\delta(t)} w_i \mathbb{1}\{\widehat{F}_\delta(t|X_i) > \widehat{F}_\delta(t|X_j)\}}{\sum_{(i,j)\in\mathcal{E}_\delta(t)} w_i}, \tag{71}$$

where

$$w_i := \frac{1}{(\widehat{S}_{\text{censor}}(Y_i))^2} \quad \text{for } i \in [n].$$

Note that $\widehat{S}_{\text{censor}}(\cdot)$ is trained the same way as we described it in Section 2.5.1: we fit a Kaplan-Meier estimator to training labels

[26] https://github.com/georgehc/survival-intro/blob/main/S6.1.4_DeepHit_competing.ipynb

$(Y_1, \mathbb{1}\{\Delta_1 = 0\}), (Y_2, \mathbb{1}\{\Delta_2 = 0\}), \ldots, (Y_n, \mathbb{1}\{\Delta_n = 0\})$. Values of $C^{\text{td}}_{\delta,t}$ are between 0 and 1, where higher is better.

We could of course integrate $C^{\text{td}}_{\delta,t}$ over time (as done in Definition 2.4) to get an integrated $C^{\text{td}}_{\delta,t}$ index for competing risks. As this is straightforward, we omit writing a formal definition. Similarly, it is straightforward to generalize the time-dependent AUC score from Section 2.5.1 to the competing risks setting as well (recall that the time-dependent AUC score we presented was just a slight modification of C^{td}_t).

Brier score. The Brier score (Definition 2.5) can also be generalized to the competing risks setting [Gerds and Kattan, 2021, Section 5.4.2]. Just as with the C^{td} and truncated time-dependent concordance indices stated above, the competing risks version of the Brier score also is specific to a single critical event $\delta \in [k]$. As with the Brier score for the standard survival analysis setup (Section 2.5.2), the competing risks version also depends on a specific time of evaluation $t \geq 0$. We have the following.

Definition 6.3 (Brier score for competing risks)**.** Let $\delta \in [k]$ and $t \geq 0$. Suppose that we have a CIF estimate $\widehat{F}_\delta(\cdot|x)$ for any $x \in \mathcal{X}$. We define the Brier score for event δ at time $t \geq 0$ by

$$\begin{aligned}\text{BS}_\delta(t) := \frac{1}{n}\sum_{i=1}^{n}\Bigg[&\frac{(1-\widehat{F}_\delta(t|X_i))^2\mathbb{1}\{\Delta_i = \delta\}\mathbb{1}\{Y_i \leq t\}}{\widehat{S}_{\text{censor}}(Y_i)} \\ &+ \frac{(\widehat{F}_\delta(t|X_i))^2\mathbb{1}\{\Delta_i \neq \delta \text{ and } \Delta_i \neq 0\}\mathbb{1}\{Y_i \leq t\}}{\widehat{S}_{\text{censor}}(Y_i)} \\ &+ \frac{(\widehat{F}_\delta(t|X_i))^2\mathbb{1}\{Y_i > t\}}{\widehat{S}_{\text{censor}}(t)}\Bigg],\end{aligned}$$

where $\widehat{S}_{\text{censor}}(\cdot)$ is trained the same way as in Section 2.5.1. Brier scores are nonnegative, where lower is better.

Of course, we can integrate the Brier score over time to obtain an integrated Brier score (as done in Definition 2.6).

6.2 Dynamic Time-to-Event Prediction with Competing Risks

We now generalize our setup from Section 6.1 to the setting where every data point is a time series. Each time series could vary in length (in terms of the number of time steps), and the time series could be irregularly sampled, meaning that the amount of time that elapses between consecutive time steps of a time series can vary. Having input data be time series can already be accommodated by the problem setup from Section 6.1! For example, using the DeepHit model, we could simply set $\mathbf{f}(\cdot;\theta)$ (equation (67)) to be a recurrent neural network (RNN), which accepts variable-length time series as inputs. Consequently, training and test data could be variable-length time series.

Thus, we reuse the exact same statistical framework for the training data as in Section 6.1.1. However, now we set the raw input space $\mathcal{X}$ in a particular manner, as we describe in Section 6.2.1. The main conceptual difference will be that we phrase the prediction task so that it depends on how much of a test time series we see. We state this new prediction task in Section 6.2.2. We then show how DeepHit can be used for this time series prediction task by choosing $\mathbf{f}(\cdot;\theta)$ based on an RNN with an attention module in Section 6.2.3. By introducing an RNN, during model training, we add an extra loss term that is meant to help the RNN learn a useful latent representation of time series (*i.e.*, we use the RNN to learn how the time series evolves over time). The resulting model is called Dynamic-DeepHit [Lee et al., 2019]. We comment on handling sequences of critical events occurring in Section 6.2.4.

A key takeaway in our exposition is that the time series version of the competing risks problem can just be reduced to the standard competing risks problem of Section 6.1. How we go from DeepHit to Dynamic-DeepHit is just in how we specify the base neural network $\mathbf{f}(\cdot;\theta)$ of DeepHit, and the addition of a loss term that learns temporal dynamics. These design steps that enable us to work with variable-length time series as inputs is *not* special to the competing risks setup and works also when the number of critical events is $k = 1$. In other words, the high-level idea of how we go from DeepHit to Dynamic-DeepHit can be applied to other deep time-to-event prediction models (such as the ones we covered in Sections 2 to 5) to enable them to work with variable-length time series as inputs.

6.2.1 Variable-length Time Series as Training Data

We denote training point i's raw input as

$$X_i := \Big(\underbrace{(U_i^{(1)}, V_i^{(1)})}_{\text{time step 1}}, \underbrace{(U_i^{(2)}, V_i^{(2)})}_{\text{time step 2}}, \ldots, \underbrace{(U_i^{(M_i)}, V_i^{(M_i)})}_{\text{time step } M_i} \Big),$$

where $M_i \in \{1, 2, \ldots\}$ is the number of time steps for point i, and at time step $m \in [M_i]$ (sorted chronologically), $U_i^{(m)} \in \mathcal{U}$ is the raw input ($\mathcal{U}$ is the raw input space for a single time step and is an input space that standard neural network software can work with), and $V_i^{(m)} \in \mathbb{R}$ is the timestamp. For example, the amount of time between time steps m and $m+1$ is $V_i^{(m+1)} - V_i^{(m)}$. A common assumption is that $V_i^{(1)} := 0$. The raw input space $\mathcal{X}$ consists of all possible time series of the format above. As our notation suggests, different training points i can have different numbers of time steps M_i. Again, RNNs readily accommodate this sort of time series data that can vary in length. (Transformer models could also be used to accept variable-length time series as inputs.)

In terms of ground truth information, just as in the Section 6.1.1, training point i has an event indicator $\Delta_i \in \{0, 1, \ldots, k\}$. If $\Delta_i = 0$, then the observed time Y_i is a censoring time. Otherwise, Δ_i is equal to the critical event that happened earliest to point i, and Y_i is the time when this critical event happened. We assume that Y_i starts measuring time starting from the last observed timestamp $V_i^{(M_i)}$. This means that at any time step $m \in [M_i]$, we are at timestamp $V_i^{(m)}$, and the time until the earliest critical event or censoring happens is $Y_i + (V_i^{(M_i)} - V_i^{(m)})$.

Even though this way of specifying the training data is a special case of the framework in Section 6.1.1, the notation we introduced here will be important when we talk about prediction next.

6.2.2 Prediction Task

We now write any test raw input x as

$$x = \Big((u^{(1)}, z^{(1)}), (u^{(2)}, z^{(2)}), \ldots \Big)$$

using the same format as the training data except where we do not pre-specify a last time step. We denote x truncated to only include its initial m time steps as

$$x^{(\leq m)} := \Big((u^{(1)}, v^{(1)}), (u^{(2)}, v^{(2)}), \ldots, (u^{(m)}, v^{(m)}) \Big).$$

As time progresses, we could see more of x, similar to what would happen in some real applications (such as a patient in a hospital intensive care unit continuously getting new measurements taken over time).

For any $m \in \{1, 2, \ldots\}$, just as in the prediction setup from Section 6.1.2, we sample nonnegative durations $T_{\text{test}} := (T_{\text{test},1}, T_{\text{test},2}, \ldots, T_{\text{test},k})$ from $\mathbb{P}_{T|X}(\cdot|x^{(\leq m)})$, and we again define

$$Y_{\text{test}} := \min\{T_{\text{test},1}, T_{\text{test},2}, \ldots, T_{\text{test},k}\}, \quad \Delta_{\text{test}} := \arg\min_{\delta \in [k]} T_{\text{test},\delta}.$$

Then we aim to predict the following dynamic version of the CIF (equation (64)) that depends on the number of time steps revealed m:

$$F_\delta(t|x,m) := \mathbb{P}(Y_{\text{test}} \leq v^{(m)} + t, \Delta_{\text{test}} = \delta \mid X = x^{(\leq m)}, Y_{\text{test}} > v^{(m)}) \\ \text{for } \delta \in [k], m \in \{1,2,\dots\}, t \geq 0, x \in \mathcal{X}.$$

This is the probability that the earliest critical event that happens is δ, and it happens within time duration t, starting from timestamp $v^{(m)}$ (the timestamp of the m-th time step). Note that the idea that we are starting the prediction from timestamp $v^{(m)}$ (so that this time step is viewed as the "origin" of the time-to-event prediction model) is not actually a limitation of this setup.[27]

The version of the CIF where we discretize duration t to only take on values along the grid $\tau_{(1)} < \tau_{(2)} < \cdots < \tau_{(L)}$ is given by

$$F_\delta[\ell|x,m] := \mathbb{P}(Y_{\text{test}} \leq v^{(m)} + \tau_{(\ell)}, \Delta_{\text{test}} = \delta \mid X = x^{(\leq m)}, Y_{\text{test}} > v^{(m)}) \\ \text{for } \delta \in [k], m \in \{1,2,\dots\}, \ell \in [L], x \in \mathcal{X}. \tag{72}$$

Importantly, we do not have to discretize the timestamps $v^{(1)}, v^{(2)}, \dots$ Even if we do discretize timestamps, how timestamps are discretized does not have to be the same way as how duration t is discretized. When timestamps are not discretized, the PMF version of equation (72) is

$$f_\delta[\ell|x,m] \\ := \mathbb{P}\Big(Y_{\text{test}} - v^{(m)} \in (\tau_{(\ell-1)}, \tau_{(\ell)}], \Delta_{\text{test}} = \delta \;\Big|\; X = x^{(\leq m)}, Y_{\text{test}} > v^{(m)}\Big) \\ \text{for } \delta \in [k], m \in \{1,2,\dots\}, \ell \in [L], x \in \mathcal{X}, \tag{73}$$

where $\tau_{(0)} := 0$. We will use this PMF in a moment.

Evaluation metrics. For this prediction task, the same evaluation metrics as in the standard competing risks setting could be used, although there would be a question of whether we care about reporting evaluation metrics as a function of how much of a test time series we see. For example, for test data, we could just predict starting from each of their final time steps, in which case using existing competing risks evaluation metrics would be straightforward. However, we could instead report evaluation metrics using, for instance, only up to the first hour of test time series. Then we report evaluation metrics using only up to the second hour of test time series, *etc*. For a concrete example of this, see Table 1 of Shen et al. [2023]. Note that here we used timestamp thresholds (*e.g.*, 1 hour, 2 hours) rather than time step thresholds (*e.g.*, 1 time step, 2 time steps) in case time steps are highly irregularly sampled across data points.

Overall, from what we can tell, how researchers evaluate models in this dynamic setting has still not become as standardized as in the regular competing risks setting and certainly not as standardized as in the standard right-censored time-to-event prediction setting of Section 2. For a recently proposed "dynamic c-index", see the paper by Putzel et al. [2021].

The standard competing risks setting as a special case. If all time series in raw input space $\mathcal{X}$ are restricted to only have one time step, then the entire problem setup would be the same as the standard competing risks setup of Section 6.1.

[27] It is important to keep in mind that how we have set up the problem, the neural network at any given time step actually knows how much time has elapsed for a data point (which is a time series). Specifically, suppose that we have observed m time steps so far of test raw input x, meaning that we have observed

$$x^{(\leq m)} = \Big((u^{(1)}, v^{(1)}), \dots, (u^{(m)}, v^{(m)})\Big).$$

The neural net is being asked to make a prediction starting at timestamp $v^{(m)}$, treating timestamp $v^{(m)}$ as the "origin". Note that the neural net also has access to $v^{(1)}$ as well. Thus, if desired, the modeler could specify the base neural network so that it explicitly depends on the difference $v^{(m)} - v^{(1)}$, meaning that the neural network knows, as one of its inputs, how much time has elapsed for the current time series since we first started observing it. This idea could be used to set up the base neural network so that it instead views timestamp $v^{(1)}$ as the "origin" rather than $v^{(m)}$.

6.2.3 Example Model: Dynamic-DeepHit

As we pointed out earlier, DeepHit can already work for this new problem setup provided that we set the neural network $\mathbf{f}(\cdot;\theta)$ in equation (67) to accept variable-length time series as inputs (in fact, the training procedure can stay the same although we will add another loss term). We provide details on how to specify $\mathbf{f}(\cdot;\theta)$ shortly. The output of $\mathbf{f}(\cdot;\theta)$ given any time series input is going to still be $L \cdot k$ numbers. We explain how to interpret these numbers first using our new time series notation. In other words, we explain what $\mathbf{f}(\cdot;\theta)$ is predicting, as this will be helpful in explaining the model architecture and training.

Prediction. Given time series $x^{(\leq m)}$, the ℓ-th row, δ-th column of $\mathbf{f}(x^{(\leq m)};\theta) \in [0,1]^{L\times k}$ is used to model $f_\delta[\ell|x,m]$ (the PMF in equation (73)). In other words, DeepHit uses $x^{(\leq m)}$ to predict CIFs for the different critical events starting from timestamp $v^{(m)}$. In more detail, the CIFs are giving probabilities of the k critical events happening within time duration $t \in \{\tau_{(1)}, \tau_{(2)}, \ldots, \tau_{(L)}\}$ starting from timestamp $v^{(m)}$.

How to specify the neural network $\mathbf{f}(\cdot;\theta)$. We give an overview of how Dynamic-DeepHit [Lee et al., 2019] specifies $\mathbf{f}(\cdot;\theta)$, deferring details to the original paper. Our exposition will be slightly more general than Lee *et al.* as we aim to convey the key high-level ideas. For example, Lee *et al.* explicitly keep track of a missingness vector per time step (that indicates which features are missing) whereas we do not include this (the missingness vector could just be included as part of the raw input space $\mathcal{U}$ for a single time step).

Given time series $x^{(\leq m)} = \big((u^{(1)}, v^{(1)}), \ldots, (u^{(m)}, v^{(m)})\big)$, Dynamic-DeepHit sets $\mathbf{f}(\cdot;\theta)$ to do the following (see accompanying Figure 7):

1. We first feed the input time series $x^{(\leq m-1)}$ (we exclude the last time step) into a user-specified RNN (with d_{hidden} output features per time step, for a user-specified number of dimensions d_{hidden}), where we slightly transform what the input looks like per time step. Specifically at time step $p \in [m-1]$ (again, the last time step is excluded), the input to the RNN is taken to be $(u^{(p)}, v^{(p+1)} - v^{(p)})$, *i.e.*, we also supply a time duration to get to the next time step. The last time step's input $u^{(m)}$ is not used with the RNN, but will be used later on. The RNN's output at time step $p \in [m-1]$ is denoted as $\widetilde{u}^{(p)} \in \mathbb{R}^{d_{\text{hidden}}}$. This first step is shown on the left side of Figure 7.

   Note that depending on the format of a single time step's input space $\mathcal{U}$, some additional neural network components may be needed (our diagram and also the original Dynamic-DeepHit treat $\mathcal{U}$ as tabular data, but this need not be the case). For example, if $\mathcal{U}$ corresponds to images of a fixed shape, then prior to feeding each $u^{(p)}$ into an RNN cell, we could first apply a convolutional neural network or a vision transformer to convert $u^{(p)}$ into a fixed-length feature vector representation that then goes into an RNN cell.

2. Next, given the RNN outputs $\widetilde{u}^{(1)}, \ldots, \widetilde{u}^{(m-1)} \in \mathbb{R}^{d_{\text{hidden}}}$ as well as the raw input $u^{(m)} \in \mathcal{U}$, we summarize all this information into a fixed-length summary vector $\widetilde{s} \in \mathbb{R}^{d_{\text{hidden}}}$. To do this, we first let $\mathbf{f}_{\text{attention}}(\cdot;\theta) : \mathbb{R}^{d_{\text{hidden}}} \times \mathcal{U} \to \mathbb{R}$ be a user-specified feed-forward neural network, such as a multilayer perceptron (MLP). Note that for $p \in [m-1]$, the output value $\mathbf{f}_{\text{attention}}\big((\widetilde{u}^{(p)}, u^{(m)});\theta\big)$ is a single number. Then we set the summary vector to be $\widetilde{s} = \sum_{p=1}^{m-1} a_p \widetilde{u}^{(p)}$, where the weights $a_1, a_2, \ldots, a_{m-1}$ are given by

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{m-1} \end{bmatrix} := \text{softmax}\left(\begin{bmatrix} \mathbf{f}_{\text{attention}}\big((\widetilde{u}^{(1)}, u^{(m)});\theta\big) \\ \mathbf{f}_{\text{attention}}\big((\widetilde{u}^{(2)}, u^{(m)});\theta\big) \\ \vdots \\ \mathbf{f}_{\text{attention}}\big((\widetilde{u}^{(m-1)}, u^{(m)});\theta\big) \end{bmatrix}\right) \in [0,1]^{m-1}.$$

   This step is labeled as the "temporal attention" block in the middle of Figure 7.

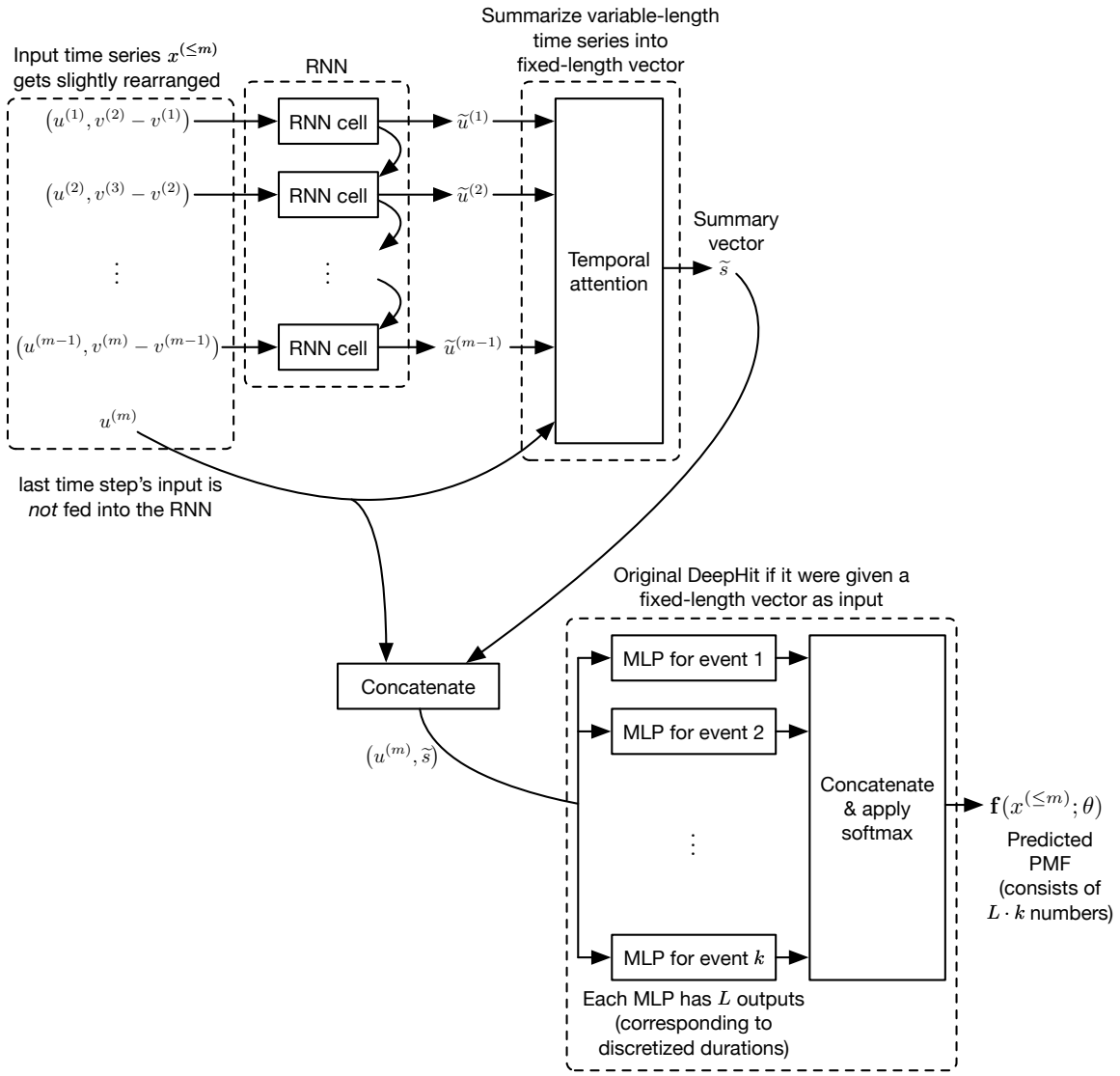


Figure 7: Dynamic-DeepHit neural network architecture.

3. We then combine the last time step's input $u^{(m)}$ with the summary vector $\widetilde{s}$ outputted by the temporal attention block to obtain the concatenated vector $(u^{(m)}, \widetilde{s})$.

4. Lastly, we treat the concatenated vector $(u^{(m)}, \widetilde{s})$ as the input to k different MLPs (one per critical event) that each outputs L numbers (corresponding to different time durations $\tau_{(1)}, \tau_{(2)}, \ldots, \tau_{(L)}$), and the overall output (across the k MLPs) is concatenated and passed through a softmax layer to produce the final neural network output $\mathbf{f}(x^{(\leq m)}; \theta)$ (the softmax enforces the constraint that the PMF sums to 1). This fourth step is shown on the bottom right of Figure 7. The final output does not have to be reshaped to be L-by-k as we had already pointed out when we first stated equation (67).

   Note that this last step uses the neural network architecture from the original DeepHit paper [Lee et al., 2018] that is meant for handling raw inputs that are fixed-length feature vectors (in how we presented DeepHit in Section 6.1.4, we intentionally stated it in a more general fashion without assuming raw inputs must be fixed-length vectors).

There is one last neural network component that is not shown in Figure 7 as it is not used to

compute the output value $\mathbf{f}(x^{(\leq m)};\theta)$. In particular, Dynamic-DeepHit also requires that at time step $p \in [m-1]$, the RNN on the left side of Figure 7 can output an estimate $\widehat{u}^{(p+1)}$ of the next time step's raw input $u^{(p+1)}$. There are different ways to achieve this. For example:

- If $\mathcal{U} = \mathbb{R}^d$, then we can choose the RNN in Figure 7 to be a type of RNN that already distinguishes between hidden state vectors and output state vectors (such as LSTMs [Hochreiter and Schmidhuber, 1997]), in which case we let the hidden state vectors be what we denoted as the $\widetilde{u}^{(p)}$ variables, and we use the output state vectors to predict the next steps' feature vectors (we would set the output state vector to consist of d entries). Thus, we could just denote these output state vectors as $\widehat{u}^{(2)}, \widehat{u}^{(3)}, \ldots, \widehat{u}^{(m)} \in \mathbb{R}^d$.

- An alternative strategy that also works if $\mathcal{U}$ is not necessarily $\mathbb{R}^d$ is that we can feed $\widetilde{u}^{(p)}$, along with the time duration to get to the next time step (*i.e.*, $v^{(p+1)} - v^{(p)}$), into a user-specified feed-forward network $\mathbf{f}_{\text{next-time-step}}(\cdot;\theta) : \mathbb{R}^{d_{\text{hidden}}} \times [0,\infty) \to \mathcal{U}$ to produce the estimate $\widehat{u}^{(p+1)}$, *i.e.*,

$$\widehat{u}^{(p+1)} := \mathbf{f}_{\text{next-time-step}}\big((\widetilde{u}^{(p)}, v^{(p+1)} - v^{(p)});\theta\big) \quad \text{for } p \in [m-1].$$

In both of these cases, we would be able to come up with estimate $\widehat{u}^{(p)}$ of $u^{(p)}$ for each $p \in \{2,3,\ldots,m\}$.

To summarize, the overall neural network $\mathbf{f}(\cdot;\theta)$ consists of an RNN, an attention network $\mathbf{f}_{\text{attention}}(\cdot;\theta)$, and MLPs for each critical event. Moreover, at the RNN stage of the neural network, we have a neural network component that predicts the next time step's raw input as described above. Note that the variable θ contains the parameters of all the neural network components involved.

Model training. Dynamic-DeepHit reuses the same training loss (equation (70)) as regular DeepHit but adds another loss term that measures how accurate each $\widehat{u}^{(p)}$ predicts $u^{(p)}$ for $p \in \{2,3,\ldots,m\}$. This new loss is

$$\mathbf{L}_{\text{next-time-step}}(\theta) := \frac{1}{n} \cdot \frac{1}{m-1} \sum_{i=1}^{n} \sum_{p=2}^{m} \zeta\big(\widehat{u}^{(p)}(\theta), u^{(p)}\big),$$

where we now emphasize that $\widehat{u}^{(p)}$ depends on the parameter variable θ, and we have a user-specified error function $\zeta : \mathcal{U} \times \mathcal{U} \to [0,\infty)$. For example, if $\mathcal{U} = \mathbb{R}^d$, then we could use squared Euclidean distance $\zeta(u,u') := \|u-u'\|^2$. The overall training loss is thus

$$\begin{aligned}&\mathbf{L}_{\text{Dynamic-DeepHit}}(\theta)\\ &\quad = \mathbf{L}_{\text{DeepHit-NLL}}(\theta) + \mathbf{L}_{\text{DeepHit-ranking}}(\theta;\eta,\sigma) + \gamma \mathbf{L}_{\text{next-time-step}}(\theta),\end{aligned}$$

where $\gamma \geq 0$ is a hyperparameter for how much to weight the new loss (as a reminder, the ranking loss already has a hyperparameter $\eta = (\eta_1,\ldots,\eta_k)$ that weights the ranking loss contributions of the different critical events). Note that for the negative log likelihood component of the loss, per training time series X_i, we consider prediction only starting at the final time step (*i.e.*, for each training point $i \in [n]$, we predict starting at time $V_i^{(M_i)}$, where the time until the earliest critical event happens is Y_i).

We provide a Jupyter notebook that implements Dynamic-DeepHit in our companion code repository.[28] This notebook builds on the DeepHit Jupyter notebook that we provided a link for in Section 6.1.4. We again use the PBC dataset but now treat the data as variable-length time series rather than first converting the dataset to be tabular.

Practical considerations. After model training, the goal is to use the model to repeatedly make predictions *as we see more and more of an individual data point's time series* (as a reminder, it is helpful to have in mind a real-time application where data keep streaming in, and we keep updating our

[28] https://github.com/georgehc/survival-intro/blob/main/S6.2.3_Dynamic-DeepHit.ipynb

predictions). During training, it is best to try to mimic the same prediction setup as what we would encounter during model testing. In other words, for the i-th training point, which is a time series of length M_i, it can be helpful to actually view this individual training point as M_i different "augmented" training points (we state each of these with its corresponding prediction task):

- Given only the first time step of this training time series (so that we observe $(U_i^{(1)}, V_i^{(1)})$), predict the CIFs starting at timestamp $V_i^{(1)}$ (*i.e.*, equation (72)), where we treat the ground truth observed time as $Y_i + (V_i^{(M_i)} - V_i^{(1)})$ and the ground truth event indicator as Δ_i.
- Given the first two time steps of this training time series (so that we observe $(U_i^{(1)}, V_i^{(1)}), (U_i^{(2)}, V_i^{(2)})$), predict the CIFs starting at timestamp $V_i^{(2)}$, where we treat the ground truth observed time as $Y_i + (V_i^{(M_i)} - V_i^{(2)})$ and the ground truth event indicator as Δ_i.
- ...
- Given all M_i observed time steps (so that we observe $(U_i^{(1)}, V_i^{(1)}), \ldots, (U_i^{(M_i)}, V_i^{(M_i)})$), predict the CIFs starting at timestamp $V_i^{(M_i)}$, where we treat the ground truth observed time as $Y_i + (V_i^{(M_i)} - V_i^{(M_i)}) = Y_i$ and the ground truth event indicator as Δ_i. (Note that this actually corresponds to the original i-th training point.)

We refer to these as augmented training points just to emphasize that these are actually constructed based on an original (or "non-augmented") training point (which is in fact just the very last augmented training point listed above). Note that as an alternative to using all M_i of these augmented training points, we could randomly choose one of them (*e.g.*, per neural network training epoch, we randomly choose a different one of the M_i augmented training points above).

Very importantly, we point out that the above augmentation strategy could be essential in practice. If one does *not* use the augmentation strategy and only uses the original training time series (so that for the i-th training point, we only use the last augmented training point listed above), then it is possible that the training procedure could, in some sense, be fooled into learning a useless model. A fundamental problem here is one of sampling bias in how the training data are collected.

As an extreme example of sampling bias, suppose that the training time series were collected in a manner where the very last time step always contains a magical feature that deterministically says what critical event happens within the next hour for the data point (but this feature is missing or not helpful at all preceding time steps). If the model is only ever trained in a manner where we always saw this very last time step for every training time series, then the model could learn to just rely on the magical feature (only at the last time step) and nothing else. *However, this model would not have been trained appropriately as to mimic making predictions for a time series as we see more of it!* Instead, if during training, the model was forced to realize that we are predicting starting from a time step that is not necessarily the last one so that the magical feature is not always helpful (note that we could ensure this by using the above augmentation strategy), then the model would be encouraged to learn to predict well even at time steps prior to the last one per training time series. Of course, if the training data were collected in a manner where the length of each training time series is random and independent of the values of the features (or raw inputs) collected over time and also independent of the survival and censoring outcomes, then we would not need to worry about this sort of sampling bias.

6.2.4 Sequences of Critical Events

The dynamic setup could readily be extended to the setting where we model a whole sequence of critical events, some of which could happen multiple times. Of course, a critical event such as death would be "terminal" so that upon encountering it, there would be no future critical events to consider. Non-terminal critical events (*e.g.*, getting admitted to a hospital) could occur multiple times though. We would simply use the same modeling strategy where the difference in interpretation is that after the earliest of k critical events happens, we can continue to make predictions! We would

just be predicting the time until the *next* critical event happens (among the k options possible). In terms of our raw inputs, we could add history information so that we keep track of what critical events have happened so far and when.

In this extension, we could think of any time series as "transitioning" between the k critical events over time, so that the critical events are *states* that a time series is moving between. This setup has been studied extensively under the name of *marked temporal point processes* (*e.g.*, Daley and Vere-Jones 2003, 2008, Du et al. 2016), sometimes written as just "marked point processes". Note that there is work in this area both with and, separately, without censoring. At this point, common modeling strategies include using attention mechanisms (*e.g.*, Zhang et al. 2020, Zuo et al. 2020) and recurrent neural networks (*e.g.*, Shchur et al. 2020) to summarize history information. Hawkes processes [Hawkes, 1971] are frequently used to model a continuous time hazard function for when a critical event will occur next. See also the neural ODE multi-state time-to-event model by Groha et al. [2020]. The SurvLatent ODE model [Moon et al., 2022], which handles the dynamic competing risks setting much like Dynamic-DeepHit but with a neural ODE, could also be adapted to this setting of reasoning about a sequence of critical events.

7 Discussion

A key goal of this monograph has been to introduce time-to-event prediction and survival analysis concepts and how they have been used with modern deep learning tools. We saw some key ideas for deriving deep survival models:

- We routinely converted between hazard, cumulative hazard, and survival functions. This amounted to using Summaries 2.1 (for continuous time) and 2.2 (for discrete time).
- Every model we covered in detail involves deriving a likelihood expression that we then turned into a negative log likelihood loss. Depending on the model, additional loss terms may be present.
- When we work with a "nonparametric" function over time, this meant that we discretized the time grid using unique times of death, and then we set the function values at the different time grid points to be parameters. *As an important practical remark: we could intentionally preprocess observed times to, for instance, discrete them into a coarse grid, even before we use a method like the Kaplan-Meier estimator. In other words, even when working with the Kaplan-Meier estimator, we could, if we wanted to, use a time grid that is not the raw unique times of death.*
- For discrete time models, the base neural network is often set so that there is an output number corresponding to each time step.
- Many evaluation metrics are ranking-based, with the idea that we can unambiguously rank many (usually not all) pairs of data points even when there is censoring. A key idea is that when the i-th point dies ($\Delta_i = 1$) and has an observed time before the j-th point (so $Y_i \leq Y_j$), then we would like the time-to-event prediction model to consider the i-th point worse off (at time Y_i) than the j-th point (also at time Y_i). The Cox model's loss function relates to ranking. DeepHit has a loss function that's directly about ranking. Including a ranking loss during model training is meant to help with ranking-based accuracy metrics.

We now also state some points that could be helpful in practice:

- For any discrete time model (*e.g.*, DeepHit, Nnet-survival, DKSA, survival kernets), we could intentionally set the time grid to be the unique times of death or, in general, based on the training data. There is no requirement that only nonparametric models are allowed to use time grids defined by the unique times of death. For instance, Chen [2024] found that sometimes using DeepHit with the time grid set to be all unique times of death resulted in time-dependent concordance indices much higher than what had previously been reported in literature.

- The use of forward filling interpolation shows up a number of times to convert a discrete time model to continuous time. In practice, we typically would not actually recommend using forward filling interpolation. Constant density or constant hazard interpolation could be used instead (among other interpolation strategies possible).
- In real applications, it is important to figure out what sort of time resolution one really needs. A major selling point of time-to-event prediction models is their ability to reason about a potentially large window of time (*e.g.*, if we really need to be precise about time and want it to be continuous, then we could use a neural ODE model like SODEN, whereas if we do not need to be too precise and discretizing time is okay, then a discrete time model could suffice). On the other hand, if one only needs to worry about very few time steps (*e.g.*, even if using a discrete time model, L is small), then there could be less of a benefit to using a full-fledged time-to-event prediction model vs just using survival stacking with an existing probabilistic binary classifier.
- When choosing a base neural network, we think it is helpful thinking about what happens when parameters of the base neural network go to 0, which we could encourage by regularizing neural network parameters (*e.g.*, by explicitly introducing a loss term or using weight decay). As a concrete example of this, suppose that we are training a DeepSurv model, and we set up a base neural network so that if all its parameters are 0, then the base neural network outputs zero. Then as we pointed out in Remark 3.2, the predicted survival function would approximate the Kaplan-Meier estimator. Thus, we have some intuition for how regularizing neural network parameters or using weight decay would impact the resulting learned model.
- Working with variable-length time series as inputs has really become easier thanks to recurrent neural networks as well as attention models. These show up in models like Dynamic-DeepHit and SurvLatent ODE. Note that while we did not explicitly cover transformers, one could easily use transformers instead of recurrent neural networks to handle variable-length time series.

We discuss a few topics that we either only glossed over earlier in the monograph, or we simply did not cover it at all. We first go over some textbook results of variants of the basic time-to-event prediction setup that we did not yet cover. Afterward, we cover more contemporary topics that are active areas of research.

7.1 More Variants of the Basic Time-to-Event Prediction Setup: Left and Interval Censoring, Truncation, and Cure Models

We now point out some standard textbook variants of the basic setup in Section 2. As will be apparent, the modifications needed to handle these variants are not difficult and do not dramatically change how we would derive time-to-event prediction models, which is why we have delayed their presentation until now.

We first talk about more kinds of censoring and also a concept called *truncation* (these could, for instance, be found in Chapter 3 of the textbook by Klein and Moeschberger [2003]). We then talk about how to address an issue that could show up in many applications where some data points will actually never experience the critical event and are thus "cured" from experiencing the critical event (a book on cure models is provided by Peng and Yu [2021]). For example, in predicting the time until convicted criminals reoffend, it could be that many of them will never reoffend.

Left and interval censoring. Thus far in the monograph, we have focused on the *right-censored* setup: for training data, when event indicator $\Delta_i = 1$, then it means that the observed time Y_i is the true survival time. If instead $\Delta_i = 0$, then Y_i is the censoring time, where the true survival time is after Y_i. For this setup, we use the likelihood function from equation (5), which we reproduce

below:

$$\begin{aligned}\mathcal{L} &:= \prod_{i=1}^{n} \{f(Y_i|X_i)^{\Delta_i} S(Y_i|X_i)^{1-\Delta_i}\} \\ &= \prod_{i=1}^{n} \left\{ f(Y_i|X_i)^{\Delta_i} \left[\int_{Y_i}^{\infty} f(u|X_i) \mathrm{d}u \right]^{1-\Delta_i} \right\}.\end{aligned}$$

As a reminder, the likelihood has a simple interpretation: when we observe the i-th point's true survival time, then the i-th point's contribution to the likelihood is $f(Y_i|X_i)$. When the i-th point is right-censored so that the true survival time is after Y_i, the point's contribution to the likelihood is

$$\int_{Y_i}^{\infty} f(u|X_i) \mathrm{d}u = \mathbb{P}(T > Y_i \mid X = X_i).$$

Instead, if the i-th point is *left-censored*, then this means that the true survival time is *before* the observed time Y_i (instead of *after* as in the right-censored case). As an example of left censoring, suppose that a data point corresponds to a patient and the critical event of interest is the time when a patient gets a disease. If a patient tests positive for a disease at time Y_i, then the actual time T_i when the patient got the disease would be at most Y_i, so that Y_i is a censoring time. When the i-th point is left-censored, the standard approach is to set its contribution to the likelihood to be

$$\int_{0}^{Y_i} f(u|X_i) \mathrm{d}u = \mathbb{P}(T < Y_i \mid X = X_i).$$

A third possibility is that the i-th point is *interval-censored*: continuing off this disease testing example, suppose now that a patient takes the test once at time $Y_i^{(1)}$, when the test turns up negative for the disease. Then some time later, the patient takes the test a second time at time $Y_i^{(2)}$, and this second test turns up positive. In this situation, we do not observe the exact time the patient got the disease, but we know that it is within the interval $[Y_i^{(1)}, Y_i^{(2)}]$. This is interval censoring. Importantly, when the i-th point is interval-censored, we assume that we observe two times $Y_i^{(1)}$ and $Y_i^{(2)}$ (with $Y_i^{(1)} < Y_i^{(2)}$) rather than only a single time, and the standard approach is to set the contribution of this point to the likelihood to be

$$\int_{Y_i^{(1)}}^{Y_i^{(2)}} f(u|X_i) \mathrm{d}u = \mathbb{P}\big(T \in [Y_i^{(1)}, Y_i^{(2)}] \mid X = X_i\big).$$

It is possible to have a setup where different points experience different censoring types, so that we observe $\Delta_i \in \{\text{not censored}, \text{right-censored}, \text{left-censored}, \text{interval-censored}\}$. Depending on the value of Δ_i, we switch between the different possible contributions to the likelihood that we mentioned above.

Truncation. Let's return to the right-censored setup (so $\Delta_i = 1$ means that we observe the true survival time, and $\Delta_i = 0$ means that we observe the censoring time) and consider a different issue that may arise. Suppose that we want to model survival times of people in a city (per person, time 0 would correspond to when they were born). To help with this task, we collect data on people from a retirement home in the city. Suppose that we can get their dates of birth, when they entered the retirement home, when they either died or were last checked up on (corresponding to the censoring event), and other features of these people (that we treat as the raw input per person). In terms of notation, for the i-th person, let's denote A_i to be the person's age at the time of entering the retirement home, and Y_i to be the age of the person at either the time of death or the time of censoring (whichever happens earlier, as in the usual right-censored setup). Thus, $Y_i > A_i$.

The issue here is that people could enter the retirement home at different ages, and they must have survived up to that point to even show up in the data. People who died earlier and never had the opportunity to enter the retirement home would not be modeled at all. Overall, what is

happening is that there is a selection bias: we are likely to see fewer people in the retirement home who are younger. This issue is called *left truncation*.

The standard way to correct for this sampling bias is to set the i-th person's contribution to the likelihood as:

$$\left[\frac{f(Y_i|X_i)}{S(A_i|X_i)}\right]^{\Delta_i}\left[\int_{Y_i}^{\infty}\frac{f(u|X_i)}{S(A_i|X_i)}\mathrm{d}u\right]^{1-\Delta_i}.$$

The basic idea is that the PDF used is instead now conditioned on surviving past age A_i (when the i-th person entered the retirement home), resulting in the denominator factors above. Roughly, $S(A_i|X_i)$ being smaller could be thought of as corresponding to a person who is less likely to have survived long enough to enter the retirement home; we up-weight this person's contribution so that the overall likelihood (across training individuals) tries to better resemble the city's population rather than only the retirement home's population.

Right and interval truncation are also possible. As the fix for these is similar, we refer the reader to Sections 3.4 and 3.5 of Klein and Moeschberger [2003] for examples of right and interval truncation in practice, and how to adjust the likelihood.

Cure models. Next, we discuss what happens if some data points will actually just never experience the critical event of interest, meaning that the population-level survival function satisfies $\lim_{t\to\infty} S_{\text{pop}}(t) > 0$. There are different ways to model this setup. We describe a simple variant of the *mixture cure model* [Boag, 1949, Xu and Peng, 2014].

Suppose that the i-th data point could possibly be "cured" of ever experiencing the critical event. Let $Z_i \in \{0, 1\}$ be a random variable indicating whether the i-th data point is cured (if $Z_i = 1$, then the i-th data point is cured). We assume that Z_i is sampled from some underlying distribution $\mathbb{P}_{Z|X}(\cdot|x)$ that is Bernoulli with probability $\omega(x) := 1 - \lim_{t\to\infty} S(t|x)$. Note that $\omega(\cdot)$ is called the *cure rate*. We do not get to observe Z_i.

Then the main modeling assumption is that

$$S(t|x) = \omega(x) + (1 - \omega(x))S_0(t|x),$$

where $S_0(\cdot|x)$ is a survival function referred to as the *latency*. The interpretation is that with probability $\omega(x)$, the survival probability is exactly 1 across time. Otherwise, the survival function is $S_0(\cdot|x)$. We could, for instance, set $\omega(\cdot)$ and $S_0(\cdot|x)$ to be neural networks, write the resulting likelihood function, and maximize the likelihood with a neural network optimizer to learn parameters. For more details on various cure models and how explicitly modeling the cure rate can be beneficial in practice, see, for instance, the paper by Ezquerro et al. [2023].

7.2 Causal Reasoning and Interventions

We have intentionally kept the scope of this monograph to time-to-event prediction and did not venture into the topics of causal reasoning or of interventions. Classically, assessing whether a treatment has an effect is based on population-level estimates. For example, Mantel [1966] established the now widely used *log-rank* statistical test to assess whether two different groups' time-to-event outcome distributions are different. If the two groups correspond to treatment and control groups that appear the "same" aside from the former receiving a treatment (*e.g.*, we randomly assign each individual to one of the two groups with equal probability), then the test would be measuring whether there is a "treatment effect". This test is done by comparing hazard function estimates of the two groups.[29] Put another way, we are comparing population-level estimates, viewing the two groups as two different populations.

Now with deep survival models as well as other machine learning survival models capable of predicting time-to-event outcomes at the *individual* data point level rather than only at the population level, causal questions we aim to address could be more fine-grain, such as estimating

[29] Even though the log-rank test pre-dates the classical Cox model [Cox, 1972], the two are intricately related: the log-rank test corresponds to looking at a Cox model with a single feature that indicates which group a data point is (*e.g.*, the feature is equal to 1 for points that received treatment and is 0 otherwise), and we are effectively checking whether the weight for this feature is 0 (which would mean that the time-to-event outcome distribution is the same between the two groups) [Harrell, 2015, Section 17.9].

individual treatment effects. For some recent work that uses deep learning, see the papers by Curth et al. [2021], Chapfuwa et al. [2021], and Nagpal et al. [2022a]. Non-deep-learning approaches are also possible (*e.g.*, Xu et al. 2023, Cui et al. 2023).

Supposing that one knew that an intervention works, there is a separate question of when to intervene, which has been recently studied by Damera Venkata and Bhattacharyya [2022]. Separately, in Section 1, we already pointed out that time-to-event modeling could be applied to the setting where the time-to-event outcome is actually a number of inventory items to stock [Huh et al., 2011]. Huh *et al.* use the Kaplan-Meier estimator to obtain a simple strategy for how to allocate inventory. This sort of strategy could be implemented by actual businesses.

7.3 Interpretability

At this point, interpretability and explainability are widely studied in the machine learning community (*e.g.*, see the book by Molnar [2022]). There are two main approaches taken. The first is to build a model that is inherently interpretable. The second is to learn an arbitrarily complex "black box" model and then come up with some sort of "explanation" of the black box. We discuss both of these in the context of deep survival models and then we separately mention a framework for visualizing any intermediate representation of any deep survival model.

Inherently interpretable deep survival models. As we had discussed in Section 3, the classical Cox model (which is a special case of a deep survival model) is straightforward to interpret. It uses the log partial hazard function $\mathbf{f}(x;\theta) := \theta^\top x$, where $\theta \in \mathbb{R}^d$ and $x \in \mathcal{X} \subseteq \mathbb{R}^d$. Thus, the values in θ tell us precisely how we weight the different features. A feature with weight 0 would be ignored by the model entirely. We could also ask for only a subset of the features to be explained by a linear model. For instance, suppose that each raw input can be written as $x = (u, v)$ where $u \in \mathbb{R}^{d_1}$ and $v \in \mathbb{R}^{d_2}$, where we only care about interpreting the model with respect to the features in u whereas the features in v are "nuisance" variables that are unrelated to u and that we do not care to make sense of. Then we could set the log partial hazard function to be

$$\mathbf{f}(x;\theta) := \psi^\top u + \mathbf{g}(z;\phi),$$

where $\psi \in \mathbb{R}^{d_1}$ is a parameter vector, $\mathbf{g}(\cdot;\phi) : \mathbb{R}^{d_2} \to \mathbb{R}$ is some user-specified neural network with parameter variable ϕ, and $\theta = (\psi, \phi)$. This partially linear Cox model with neural network $\mathbf{g}(\cdot;\phi)$ has known statistical guarantees [Zhong et al., 2022].

Meanwhile, in Section 4, we already discussed how deep kernel survival analysis [Chen, 2020] and survival kernets [Chen, 2024] are in some sense interpretable. As a reminder, both provide "forecast evidence": deep kernel survival analysis can tell us which training points contribute to predictions, and survival kernets can tell us which exemplar clusters contribute to predictions. For the latter, we also showed how to make visualizations like those in Figures 5 and 6. In the survival kernets paper, Chen [2024] noted that an interesting direction for future research is determining whether there are better ways to do exemplar-based clustering that, for instance, somehow improves model accuracy or model interpretability (how to define the latter of course is not straightforward). The choice of using ε-net clustering was to make the model amenable to theoretical analysis.

A number of other deep survival models have been developed that "bake in" some sort of interpretable component. For example, Chapfuwa et al. [2020], Nagpal et al. [2021b], Manduchi et al. [2022], and Jeanselme et al. [2022] also use clustering models, where clusters are learned in a supervised fashion to help with predicting a time-to-event outcome. Meanwhile, Chen et al. [2024] propose using neural topic models that are (like the clustering models just mentioned) also supervised so that the topics help with time-to-event prediction.

More recently, Sun and Qiu [2023] proposed a deep learning approach for training survival trees that are interpretable. We point out that there are also non-deep-learning based methods for training survival trees (*e.g.*, Ishwaran et al. 2008, Bertsimas et al. 2022, Zhang et al. 2024), for which so long as a learned tree does not have too many leaves, then model interpretation is straightforward. If one uses an ensemble of trees, then if we want the overall model to be straightforward to interpret, then we would have to avoid using too many trees.

Explaining black box models. As for training an arbitrarily complex deep survival model that is not interpretable and then applying a post hoc explanation tool afterward, here we begin by saying that standard tools that were not originally designed for time-to-event prediction can be used. Specifically, LIME [Ribeiro et al., 2016] and SHAP [Lundberg and Lee, 2017] have already been applied to time-to-event prediction models [Kovalev et al., 2020, Krzyziński et al., 2023].

We point out two other relatively straightforward strategies that can be applied:

- We could first train a deep survival model that is not interpretable. After training this model, we take its base neural network (with its learned parameters), and modify the last layer or a few of the last layers so that the modified base neural network can work with one of the inherently interpretable deep survival models (*e.g.*, survival kernets), at which point, we could fine-tune the modified base neural network using the loss function of the interpretable model.
- As an alternative, we could—just as before—first train a deep survival model that is not interpretable. Let's denote its resulting predicted survival function as $\widehat{S}_{\text{uninterpretable}}(\cdot|x)$. We then train one of the interpretable survival models where instead of (or in additional to) its usual loss function, we use a loss function that tries to match the interpretable model's predicted survival function with $\widehat{S}_{\text{uninterpretable}}(\cdot|x)$. Basically, we try to match the model outputs. We could of course try matching on the hazard function or the cumulative hazard function instead.

How to get these ideas to work well for deep survival models would be an interesting direction for future research.

Visualizing an intermediate representation. To try to make sense of an intermediate embedding representation of any neural network, a standard approach is to apply t-SNE [Van der Maaten and Hinton, 2008] to this embedding representation. Note that t-SNE is *not* designed to explain black box models. However, it can help us understand what semantics are captured by the embedding space (*e.g.*, by showing which raw inputs tend to map close to each other in the embedding space).

In a similar vein, Chen [2023] provided a general framework for visualizing any intermediate embedding representation used by any already trained deep survival model. The framework first identifies so-called *anchor directions* in the embedding space. Afterward, visualizations can be made to relate each anchor direction to raw features and, separately, to time-to-event outcome distributions. In a healthcare dataset where data points are patients, an anchor direction could, for instance, capture the concept of age. Naturally, younger and older patients would have different distributions for how much longer they will live. Chen's framework also comes with statistical tests that could be run to check whether an anchor direction is associated with specific raw features.

7.4 Fairness

The machine learning community has now been working on fairness quite extensively. Fairness metrics for time-to-event prediction have only recently been defined [Keya et al., 2021, Rahman and Purushotham, 2022, Zhang and Weiss, 2022]. Roughly, the key ideas of these metrics ask for: (i) similar data points to have similar predicted time-to-event outcomes, (ii) data points from different pre-defined subpopulations to have similar predicted outcomes, or (iii) data points from different pre-defined subpopulations to have similar prediction accuracy.

We point out that some of these metrics do not always make sense depending on the application. For example, in healthcare, age is often highly predictive of various time-to-event outcomes, such as the classical example of time until death. Suppose that we want a time-to-event prediction model to be "fair" across age groups. For simplicity, let's say that we just use two age groups (*e.g.*, thresholding based on whether a person is at least 65 years old). Then asking for young people and elderly people to have similar predicted times until death would not make sense. In this case, asking for the model to be equally accurate for the two subpopulations is a better notion of fairness.

In terms of encouraging fairness, the standard approach is simply to add regularization terms [Keya et al., 2021, Rahman and Purushotham, 2022, Do et al., 2023], which is typically done in practice with the knowledge of which subpopulations we want to account for and which specific

fairness metric we care about. However, enumerating all the subpopulations that we want to be fair across is not always straightforward. Minority subpopulations that might be at risk of being treated unfairly by a machine learning model could be defined by the intersection of a variety of different criteria [Buolamwini and Gebru, 2018]. For example, even just considering a few attributes commonly treated as sensitive such as age, gender, and race, we run into the issue that there are many combinations of these three that are possible, in part because we also need to decide on how precisely to discretize age.

It turns out that it is possible to use a training loss function that does *not* require the modeler to enumerate the subpopulations that we want to be fair across, and that still encourages fairness. In particular, Hu and Chen [2024] introduced a general strategy for converting a wide range of time-to-event prediction models into ones that encourage fairness using *distributionally robust optimization* (DRO) (*e.g.*, Hashimoto et al. 2018, Duchi and Namkoong 2021, Li et al. 2021, Duchi et al. 2022). Roughly, DRO minimizes the worst-case error over *all* subpopulations that are large enough (occurring with at least some user-specified probability threshold $\pi_{\min}$), and can be solved tractably. In particular, as $\pi_{\min}$ could be thought of as how "rare" of a minority subpopulation that we want the trained model to account for. As $\pi_{\min} \to 0$, we would be training the time-to-event prediction model to have the worst-case error for even an individual data point (which could be an outlier) to be as low as possible. As $\pi_{\min} \to 1$, we switch to simply carrying about the equally weighted average loss across all training data points, disregarding any sort of concern of minority subpopulations.

The technical complication to applying DRO to time-to-event prediction is that existing DRO theory uses a training loss function that decomposes across contributions of individual data points, *i.e.*, any term that shows up in the loss function depends only on a single training point. This decomposition does not hold for many deep survival models' loss functions, such as those of semi-parametric proportional hazards models (*e.g.*, the Cox model, DeepSurv, Cox-Time), DeepHit, deep kernel survival analysis, and survival kernets. Hu and Chen [2024] address this technical hurdle using a sample splitting DRO strategy, which they also established some theory for. Specifically for classical and deep Cox models, Hu and Chen also derived an exact DRO Cox approach that does not require sample splitting.

7.5 Statistical Guarantees

We have tried sprinkling known results of statistical accuracy guarantees as we progressed through the monograph. In this section, we comment more on these accuracy guarantees for deep survival models. We also briefly mention guarantees in terms of producing so-called *prediction intervals* for survival times (*e.g.*, for test patient Alice, we predict that Alice's hospital length of stay is in the interval [0.5, 2.5] days with probability at least 90%).

Accuracy guarantees. At present, the vast majority of deep survival models have no guarantees whatsoever. Even for the ones that do have guarantees, once we look closely at the fine print as to what the assumptions are, the assumptions made could be impractical.

For example, the theory for deep extended hazard models [Zhong et al., 2021] and deep partially linear Cox models [Zhong et al., 2022] make strong assumptions on the base neural networks used (*e.g.*, on the architecture and on some notion of how large the neural network parameters are), and their theory relies on the neural network optimizer reaching the global minimum of the loss function. The theory for these models also makes some restrictive assumptions on the hazard function (after all, a deep extended hazard model's hazard function still has to satisfy a specific factorization, and the deep partially linear Cox model has an even more restrictive hazard function formulation).

Meanwhile, the theory for survival kernets [Chen, 2024] is stated in terms of the intermediate embedding space and how it relates to true survival time and censoring time distributions. In more detail, the theory treats the neural network $\mathbf{f}(\cdot;\theta) : \mathcal{X} \to \mathbb{R}^{d_{\text{emb}}}$ that maps raw inputs to the embedding space as a black box so that it could be trained however the user wants. Instead, the theory requires the embedding space to satisfy "nice" properties. Some of these properties

are straightforward to enforce or encourage, such as controlling the geometry of the embedding space and its probability distribution (the running example Chen gives is constraining the output of $\mathbf{f}(\cdot;\theta)$ to be on a hypersphere, for which there are known ways of encouraging the embedding vectors to be uniform over the hypersphere [Wang and Isola, 2020, Liu et al., 2021]). However, there is no known practical method to our knowledge of verifying the conditions on how the embedding space relates to true survival and censoring time distributions. Even if instead the assumptions were on relating the raw input space (rather than the embedding space) to the true survival and censoring time distributions, we would still not have a practical way to verify such conditions.

Suffice it to say, we think that there is a lot of space for improving our theoretical understanding of deep survival models. Perhaps making some structural assumptions on the data would be helpful, such as assuming the data to come from a few clusters.

Prediction interval guarantees. If we can predict a survival function for any data point, then we could also back out a survival time point estimate (e.g., look at when the survival function crosses 1/2 to estimate the median survival time). However, is there any way to estimate error bars for these survival time point estimates? When there is no censoring, this problem has at this point been studied for a long time, where a solution that has become popular is called *conformal prediction* (early work was done by Vovk et al. [2005]; for an excellent tutorial on the topic, see the monograph by Angelopoulos and Bates [2023]).

To give a sense of how conformal prediction works, we describe an approach called *split conformal prediction* [Papadopoulos et al., 2002, Lei et al., 2015], which we intentionally phrase in terms of the survival analysis setup of Section 2 except where there is no censoring (so that we are looking at a standard regression problem). With the notation we have been using throughout the monograph but now dropping event indicator Δ_i variables (since they are all equal to 1), we take the i.i.d. training data to be $(X_1, Y_1), \ldots, (X_n, Y_n)$, where X_i's are raw inputs and Y_i's are observed times. However, since every observed time Y_i is actually a survival time T_i, for clarity, we write that the training data are $(X_1, T_1), \ldots, (X_n, T_n)$. (Using this notation will be helpful as we reintroduce censoring later.) We separately suppose that we have access to n_+ so-called "calibration" data points $(X_1^+, T_1^+), \ldots, (X_{n_+}^+, T_{n_+}^+)$ that are i.i.d. with the same distribution as the training data. Importantly, the calibration data do *not* serve the same purpose as validation data (which are used to help tune hyperparameters; this would be considered as part of the training procedure). The calibration data should not be seen by the training procedure whatsoever.

Again, an example of a survival time prediction interval could be [0.5, 2.5] days. This interval would hold with some probability (since we typically cannot be absolutely certain of survival time prediction intervals unless it is a trivial interval like $[0, \infty)$). Let $\alpha \in (0, 1)$ be a user-specified probability tolerance, where we aim to construct a prediction interval that holds with probability at least $1 - \alpha$ (so that if we want to construct prediction intervals that hold with probability at least 90%, we would pick $\alpha = 0.1$).

Then split conformal prediction works as follows to produce survival time prediction intervals given the user-specified value for α:

1. We train any regression model of our choosing on the training data $(X_1, T_1), \ldots, (X_n, T_n)$ to produce a survival time estimator $\widehat{T} : \mathcal{X} \to [0, \infty)$, meaning that $\widehat{T}(x)$ is the predicted survival time for any raw input $x \in \mathcal{X}$.

2. We compute residuals for the calibration data: $R_i = |T_i^+ - \widehat{T}(X_i^+)|$ for $i = 1, 2, \ldots, n_+$. We also introduce an additional residual $R_{n_+ + 1} := \infty$.

3. Note that the residuals $R_1, \ldots, R_{n_+ + 1}$ can be sorted (even when we have $R_{n_+ + 1} = \infty$). Let $\widehat{q}$ be the $(1 - \alpha)$-th quantile of this empirical distribution. Put another way, if we denoted the sorted residuals as $R_{(1)} \leq R_{(2)} \leq \cdots \leq R_{(n_+ + 1)} = \infty$ (breaking ties randomly), then $\widehat{q} := R_{(\lceil (1-\alpha)(n_+ + 1) \rceil)}$. (For example, if $\alpha = 0.1$, then we would be setting $\widehat{q}$ to be the 90 percentile value of $R_1, \ldots, R_{n_+ + 1}$.)

4. For any $x \in \mathcal{X}$, we output the prediction interval for x to be $\widehat{\mathcal{C}}(x) := [\widehat{T}(x) - \widehat{q}, \widehat{T}(x) + \widehat{q}]$. Thus, for any test point $x \in \mathcal{X}$, our regression model's survival time prediction for x is $\widehat{T}(x)$,

and we estimate the error bar to be $\widehat{q}$ (this error bar does not depend on x).

A major theoretical result for split conformal prediction is that the prediction intervals it produces are "statistically valid". In particular, if we were to sample a data point (X, T) from the same distribution underlying the training (and calibration) data, then with probability at least $1 - \alpha$, the true survival time T will be in the prediction interval $\mathcal{C}(X)$ (see Theorem 2.2 of Lei et al. [2018]).

Now let's discuss what happens when there is censoring. The training data are now $(X_1, Y_1, \Delta_1), \ldots, (X_n, Y_n, \Delta_n)$, and the calibration data are $(X_1^\dagger, Y_1^\dagger, \Delta_1^\dagger), \ldots, (X_{n_\dagger}^\dagger, Y_{n_\dagger}^\dagger, \Delta_{n_\dagger}^\dagger)$—again, we assume all these data points to be i.i.d. The key observation for how to handle censoring is actually immediate given our discussion in Section 2.5.4. When there is censoring, the above split conformal prediction procedure can still be used but with some minor changes:

- In step 1, we would learn a survival model using the training data. We use this survival model to predict a survival time (again, the survival model chosen might actually predict a survival function from which we back out a median survival time estimate). We still denote this survival time estimator as $\widehat{T} : \mathcal{X} \to [0, \infty)$.
- In step 2, we do not have the ground truth survival time for censored calibration points, but we could still compute residuals when there is censoring. For example, close to the start of Section 2.5.4, we mentioned that a naive residual that could be computed is called the hinge error; the absolute error version would be

$$R_i = \begin{cases} |Y_i^\dagger - \widehat{T}(X_i^\dagger)| & \text{if } \Delta_i^\dagger = 1, \\ (Y_i^\dagger - \widehat{T}(X_i^\dagger))\mathbb{1}\{Y_i > \widehat{T}(X_i^\dagger)\} & \text{if } \Delta_i^\dagger = 0. \end{cases}$$

  Specifically, when a calibration point is censored ($\Delta_i^\dagger = 0$), the residual is nonnegative only when the predicted survival time is less than the observed time (which is known to be a censoring time), and when $Y_i \leq \widehat{T}(X_i^\dagger)$, we optimistically assume that the error is 0 (even though it could be that the predicted survival time, for instance, way overestimates the unknown true survival time).

  A different choice of residual is to use each individual data point's error from equation (38), which—phrased in the context of calibration data—would be written as

$$R_i = \Delta_i^\dagger |\widehat{T}(X_i^\dagger) - Y_i| + (1 - \Delta_i)|\widehat{T}(X_i^\dagger) - T_i^{\text{PO}}|,$$

  where T_i^{PO} is defined in equation (37) (we would take the evaluation data points to be the calibration set). Of course, rather than using the pseudo observation approach to impute the unknown true survival time, we could use some other imputation approach (such as the margin method described in Section 2.5.4).

Once we have made the above changes, the rest of the split conformal prediction procedure stays the same, and the upshot is that we can compute prediction intervals for survival times even when there is censoring. It is possible to come up with statistical guarantees for the resulting prediction intervals, much like how there are guarantees for conformal prediction in the standard regression setting without censoring. Existing such results have been established by Chen [2020] and Candès et al. [2023]. Also, it is possible to use conformal prediction to make a survival model better calibrated [Qi et al., 2024b].

7.6 Empirical Evaluation

At the time of writing, there is no comprehensive, thorough empirical comparison of different survival models (regardless of whether they use deep learning or not) on a large selection of survival analysis datasets. Understandably, this is an onerous task. Many models have quite a few hyperparameters to tune, and figuring out the "right" range of values to try across datasets could be challenging. Also, the availability of publicly available standard datasets is, at present, not remotely at the same level as for classification and regression.

In conducting this sort of benchmark, we think that it is important to try to control for differences that are not related to the actual time-to-event prediction model. For example, if we were to compare two different deep survival models but we supplied them with vastly different base neural networks, then the differences in how the two models perform might be explained more by the differences in the base neural networks than by differences in the two survival models' modeling assumptions.[30] As another example, in conducting minibatch gradient descent to train two different deep survival models, whether we use early stopping (*e.g.*, if there is no improvement in some validation set evaluation score after 10 training epochs, then stop training and backtrack to using whichever epoch's model parameters achieved the highest validation set evaluation score) should be the same across models.[31]

Meanwhile, devising new evaluation metrics for survival models remains an active area of research (see, for instance, the recent paper by Qi et al. [2023]). The community has focused a lot on ranking-based accuracy metrics, such as time-dependent concordance indices. While these could be useful for ranking data points such as patients in a clinical task (*e.g.*, to help hospitals prioritize which patients to focus on), ranking is not always what we want to do. In some cases, having an actual predicted value of the time-to-event outcome could be useful, in which case, the survival time error metrics in Section 2.5.4 could be useful (*e.g.*, MAE measured against ground truth survival times for uncensored cases and against imputed survival times for censored cases). However, these error metrics based on pointwise estimates of survival times do not easily generalize to the setting where there are multiple competing events (or also to the setting of cure models mentioned in Section 7.1), where the problem is that for a specific critical event, the ground truth could be that this event never happens for a data point.

Separately, it does not help that various metrics require an estimate of the censoring time distribution $\mathbb{P}_C$ (in terms of the function $S_{\text{censor}}(t) = 1 - \mathbb{P}_C(t)$). We had pointed out that when censoring times are independent of the raw inputs, then it suffices to estimate S_{censor} using the Kaplan-Meier estimator. When this independence assumption does not hold, then there exist versions of some of the evaluation metrics we mentioned that would still work if we instead have a good estimate of the distribution $\mathbb{P}_{C|X}$, but this distribution could be as difficult to estimate as the target distribution we are trying to predict $\mathbb{P}_{T|X}$. Regardless, a bad estimate of the censoring time distribution could cause evaluation metrics that depend on this distribution to be unreliable.

Turning toward the dynamic setup of Section 6.2 where we see more of a time series over time and can continually make predictions, it is unclear what evaluation metrics make the most sense here. In practice, we are unlikely to need to make predictions after every single time step of new information is collected. When a prediction might actually be useful could depend on whether there is an intervention that might make sense to attempt in the near future. Meanwhile, in practice, it is likely the case that among k critical events, some are more concerning than others so that we would not want to treat all k events equally.

7.7 Large Language Models and Foundation Models

Lastly, during the writing of this monograph, large language models (LLMs) and—more generally—foundation models took over the machine learning community by storm [Bommasani et al., 2021]. These developments quickly transferred over to the survival analysis setting. After all, at a conceptual level, one could simply set the base neural network to be a foundation model.

There is already a review of how LLMs are used for survival analysis [Jeanselme et al., 2024].

[30] Of course, commonly the base neural network cannot be made identical across different time-to-event prediction models (*e.g.*, DeepSurv requires its base neural network to output a single value whereas DeepHit instead asks for many output values, one per discretized time index), but we can still have the base neural networks be as similar as possible (*e.g.*, using multilayer perceptrons that are identical except for the final layer).

[31] Some models might work better with different learning rates or different numbers of training epochs. We could try to tune these in a similar fashion across models. For example, we can fix some maximum number of training epochs across all models and then use early stopping to automatically tune on the number of training epochs. As for the learning rate, we could pre-specify a grid of learning rates that we try across models. Then per model, we use whichever learning rate achieves the best validation set evaluation score (per learning rate, we use early stopping to decide on the number of training epochs).

As is, in Section 7.6, we mentioned that there is no comprehensive experimental benchmark for survival analysis. This is also true for LLMs applied to survival analysis, where differences in experimental setups make "apples-to-apples" comparisons between LLM approaches difficult. To address this issue, in their review, Jeanselme *et al.* propose a framework for evaluating LLMs for survival analysis that aims to standardize various steps of the machine learning pipeline.

Meanwhile, specifically for electronic health records, a foundation model called MOTOR for predicting time durations until the next critical event happens has recently been published [Steinberg et al., 2024]. The underlying survival model Steinberg *et al.* used as the prediction head has been available for a number of years: a piecewise exponential model for hazard functions [Fornili et al., 2014]. The base neural network is a transformer. The novelty in the research was in scaling the resulting model to a sizable amount of data (pretraining uses 55M patient records with 9B clinical events), and demonstrating impressive transfer learning results (19 tasks across 3 healthcare datasets). Could using a different survival model as the prediction head have improved their results? What about a different base neural network? What about dramatically more data? For what kinds of time-to-event prediction problems would foundation models be most beneficial, and for what kinds would they be least beneficial—not limited to only the healthcare space? Suffice it to say, there are many open questions related to how best to use foundation models for time-to-event prediction.

Acknowledgments

I would like to first thank Jeremy Weiss for introducing me to the topic of survival analysis back when I first started as faculty at Carnegie Mellon University. This monograph is also in some sense a successor to a tutorial Jeremy and I taught on survival analysis at the Conference on Health, Inference, and Learning back in 2020. I would also like to thank a number of collaborators who I have worked on survival analysis problems with aside from Jeremy: Cheng Cheng, Amanda Coston, Jonathan Elmer, Shu Hu, Lihong Li, Zack Lipton, Xiaobin Shen, Helen Zhou, and Ren Zuo. Finally, I would like to thank the editors and reviewers at Foundation and Trends in Machine Learning for providing very helpful feedback. This work is supported by NSF CAREER award #2047981.

References

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

O. O. Aalen and S. Johansen. An empirical transition matrix for non-homogeneous markov chains based on censored observations. *Scandinavian Journal of Statistics*, pages 141–150, 1978.

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

P. D. Allison. Discrete-time methods for the analysis of event histories. *Sociological Methodology*, 13: 61–98, 1982.

A. N. Angelopoulos and S. Bates. A gentle introduction to conformal prediction and distribution-free uncertainty quantification. *Foundations and Trends® in Machine Learning*, 16(4):494–591, 2023.

L. Antolini, P. Boracchi, and E. Biganzoli. A time-dependent discrimination index for survival data. *Statistics in Medicine*, 24(24):3927–3944, 2005.

A. Avati, T. Duan, S. Zhou, K. Jung, N. H. Shah, and A. Y. Ng. Countdown regression: Sharp and calibrated survival predictions. In *Uncertainty in Artificial Intelligence*, pages 145–155. PMLR, 2020.

N. Bacaër. *A Short History of Mathematical Population Dynamics*, volume 618. Springer, 2011.

R. Beran. Nonparametric regression with randomly censored survival data. *Technical report, University of California, Berkeley*, 1981.

D. Bertsimas, J. Dunn, E. Gibson, and A. Orfanoudaki. Optimal survival trees. *Machine Learning*, 111(8):2951–3023, 2022.

P. Blanche, A. Latouche, and V. Viallon. Time-dependent auc with right-censored data: a survey. *Risk Assessment and Evaluation of Predictions*, pages 239–251, 2013.

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

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

J. W. Boag. Maximum likelihood estimates of the proportion of patients cured by cancer therapy. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 11(1):15–53, 1949.

P. E. Böhmer. Theorie der unabhängigen wahrscheinlichkeiten. In *Rapports Memoires et Proces verbaux de Septieme Congres International dActuaires Amsterdam*, volume 2, pages 327–343, 1912.

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. Chatterji, A. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec,

I. Levent, X. L. Li, X. Li, T. Ma, A. Malik, C. D. Manning, S. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. Roohani, C. Ruiz, J. Ryan, C. Ré, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

J. M. Box-Steffensmeier and B. S. Jones. *Event History Modeling: A Guide for Social Scientists*. Cambridge University Press, 2004.

J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`.

N. Breslow. Discussion of the paper by D R Cox (1972). *Journal of the Royal Statistical Society, Series B*. 34(2):216–217, 1972.

C. C. Brown. On the use of indicator variables for studying the time-dependence of parameters in a response-time model. *Biometrics*, 31(4):863–872, 1975.

J. Buolamwini and T. Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on Fairness, Accountability and Transparency*, pages 77–91. PMLR, 2018.

E. Candès, L. Lei, and Z. Ren. Conformalized survival analysis. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 85(1):24–45, 2023.

G. Chagny and A. Roche. Adaptive and minimax estimation of the cumulative distribution function given a functional covariate. *Electronic Journal of Statistics*, 8(2):2352–2404, 2014.

O. Chapelle. Modeling delayed feedback in display advertising. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1097–1105, 2014.

P. Chapfuwa, C. Tao, C. Li, C. Page, B. Goldstein, L. C. Duke, and R. Henao. Adversarial time-to-event modeling. In *International Conference on Machine Learning*, pages 735–744. PMLR, 2018.

P. Chapfuwa, C. Li, N. Mehta, L. Carin, and R. Henao. Survival cluster analysis. In *Proceedings of the ACM Conference on Health, Inference, and Learning*, pages 60–68, 2020.

P. Chapfuwa, S. Assaad, S. Zeng, M. J. Pencina, L. Carin, and R. Henao. Enabling counterfactual survival analysis with balanced representations. In *Proceedings of the Conference on Health, Inference, and Learning*, pages 133–145, 2021.

G. H. Chen. Nearest neighbor and kernel survival analysis: Nonasymptotic error bounds and strong consistency rates. In *International Conference on Machine Learning*, pages 1001–1010. PMLR, 2019.

G. H. Chen. Deep kernel survival analysis and subject-specific survival time prediction intervals. In *Machine Learning for Healthcare Conference*, pages 537–565. PMLR, 2020.

G. H. Chen. A general framework for visualizing embedding spaces of neural survival analysis models based on angular information. In *Conference on Health, Inference, and Learning*, pages 440–476. PMLR, 2023.

G. H. Chen. Survival kernets: Scalable and interpretable deep kernel survival analysis with an accuracy guarantee. *Journal of Machine Learning Research*, 25(40):1–78, 2024.

G. H. Chen and D. Shah. Explaining the success of nearest neighbor methods in prediction. *Foundations and Trends® in Machine Learning*, 10(5-6):337–588, 2018.

G. H. Chen, L. Li, R. Zuo, A. Coston, and J. C. Weiss. Neural topic models with survival supervision: Jointly predicting time-to-event outcomes and learning how clinical features relate. *Artificial Intelligence in Medicine*, 2024.

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018.

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

P. Chilinski and R. Silva. Neural likelihoods via cumulative distribution functions. In *Conference on Uncertainty in Artificial Intelligence*, pages 420–429. PMLR, 2020.

C.-F. Chung, P. Schmidt, and A. D. Witte. Survival analysis: A survey. *Journal of Quantitative Criminology*, 7:59–98, 1991.

A. Ciampi, R. S. Bush, M. Gospodarowicz, and J. E. Till. An approach to classifying prognostic factors related to survival experience for non-hodgkin's lymphoma patients: Based on a series of 982 patients: 1967–1975. *Cancer*, 47(3):621–627, 1981.

D. Collett. *Modelling Survival Data in Medical Research, Fourth Edition*. Chapman and Hall/CRC, 2023.

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

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

D. R. Cox and D. Oakes. *Analysis of Survival Data*. CRC press, 1984.

E. Craig, C. Zhong, and R. Tibshirani. Survival stacking: casting survival analysis as a classification problem. *arXiv preprint arXiv:2107.13480*, 2021.

Y. Cui, M. R. Kosorok, E. Sverdrup, S. Wager, and R. Zhu. Estimating heterogeneous treatment effects with right-censored data via causal survival forests. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 85(2):179–211, 2023.

A. Curth, C. Lee, and M. van der Schaar. SurvITE: Learning heterogeneous treatment effects from time-to-event data. In *Advances in Neural Information Processing Systems*, 2021.

D. J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes: Volume I: Elementary Theory and Methods*. Springer, 2003.

D. J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes: Volume II: General Theory and Structure*. Springer, 2008.

N. Damera Venkata and C. Bhattacharyya. When to intervene: Learning optimal intervention policies for critical events. In *Advances in Neural Information Processing Systems*, 2022.

D. Danks and C. Yau. Derivative-based neural modelling of cumulative distribution functions for survival analysis. In *International Conference on Artificial Intelligence and Statistics*, pages 7240–7256. PMLR, 2022.

C. Davidson-Pilon. lifelines: survival analysis in Python. *Journal of Open Source Software*, 4(40):1317, 2019.

H. Do, Y. Chang, Y. S. Cho, P. Smyth, and J. Zhong. Fair survival time prediction via mutual information minimization. In *Machine Learning for Healthcare Conference*, 2023.

A. B. Downey. *Think stats*. O'Reilly Media, Inc., 2011.

N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song. Recurrent marked temporal point processes: Embedding event history to vector. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1555–1564, 2016.

J. Duchi, T. Hashimoto, and H. Namkoong. Distributionally robust losses for latent covariate mixtures. *Operations Research*, 2022.

J. C. Duchi and H. Namkoong. Learning models with uniform performance via distributionally robust optimization. *The Annals of Statistics*, 49(3):1378–1406, 2021.

R. Dybowski and V. Gant. *Clinical Applications of Artificial Neural Networks*. Cambridge University Press, 2001.

C. E. Ebeling. *An Introduction to Reliability and Maintainability Engineering*. Waveland Press, 2019.

A. Ezquerro, B. Cancela, and A. López-Cheda. On the reliability of machine learning models for survival analysis when cure is a possibility. *Mathematics*, 11(19):4150, 2023.

D. Faraggi and R. Simon. A neural network model for survival data. *Statistics in Medicine*, 14:73–82, 1995.

J. P. Fine and R. J. Gray. A proportional hazards model for the subdistribution of a competing risk. *Journal of the American Statistical Association*, 94(446):496–509, 1999.

T. R. Fleming and D. P. Harrington. *Counting Processes and Survival Analysis*. John Wiley & Sons, 1991.

J. A. Foekens, H. A. Peters, M. P. Look, H. Portengen, M. Schmitt, M. D. Kramer, N. Brünner, F. Jänicke, M. E. Meijer-van Gelder, S. C. Henzen-Logmans, W. L. J. van Putten, and J. G. M. Klijn. The urokinase system of plasminogen activation and prognosis in 2780 breast cancer patients. *Cancer Research*, 60(3):636–643, 2000.

A. Földes and L. Rejtö. Strong uniform consistency for nonparametric survival curve estimators from randomly censored data. *The Annals of Statistics*, 9(1):122–129, 1981.

M. Fornili, F. Ambrogi, P. Boracchi, and E. Biganzoli. Piecewise exponential artificial neural networks (PEANN) for modeling hazard function with right censored data. In *Computational Intelligence Methods for Bioinformatics and Biostatistics: 10th International Meeting, CIBB 2013, Nice, France, June 20-22, 2013, Revised Selected Papers 10*, pages 125–136. Springer, 2014.

S. Fotso. Deep neural networks for survival analysis based on a multi-task framework. *arXiv preprint arXiv:1801.05512*, 2018.

S. Fotso et al. PySurvival: Open source package for survival analysis modeling, 2019. URL `https://www.pysurvival.io/`.

M. F. Gensheimer and B. Narasimhan. A scalable discrete-time survival model for neural networks. *PeerJ*, 7:e6257, 2019.

T. A. Gerds and M. W. Kattan. *Medical Risk Prediction Models: With Ties to Machine Learning*. Chapman and Hall/CRC, 2021.

D. V. Glass. John Graunt and his Natural and political observations. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 159(974):2–37, 1963.

T. Gneiting and A. E. Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477):359–378, 2007.

M. Goldstein, X. Han, A. Puli, A. Perotte, and R. Ranganath. X-CAL: Explicit calibration for survival analysis. In *Advances in Neural Information Processing Systems*, 2020.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

L. Gordon and R. A. Olshen. Tree-structured survival analysis. *Cancer Treatment Reports*, 69(10): 1065–1069, 1985.

E. Graf. *Explained variation measures for survival data*. PhD thesis, University of Freiburg (in German), 1998.

E. Graf, C. Schmoor, W. Sauerbrei, and M. Schumacher. Assessment and comparison of prognostic classification schemes for survival data. *Statistics in Medicine*, 18(17-18):2529–2545, 1999.

J. Graunt. Natural and political observations mentioned in a following index and made upon the bills of mortality, 1662.

R. J. Gray. A class of *K*-sample tests for comparing the cumulative incidence of a competing risk. *The Annals of Statistics*, pages 1141–1154, 1988.

S. Groha, S. M. Schmon, and A. Gusev. A general framework for survival analysis and multi-state modelling. *arXiv preprint arXiv:2006.04893*, 2020.

H. Haider, B. Hoehn, S. Davis, and R. Greiner. Effective ways to build and evaluate individual survival distributions. *Journal of Machine Learning Research*, 21(85):1–63, 2020.

E. Halley. An estimate of the degrees of the mortality of mankind; drawn from curious tables of the births and funerals at the city of breslaw; with an attempt to ascertain the price of annuities upon lives. *Philosophical Transactions of the Royal Society of London*, 17:596–610, 1693.

F. E. Harrell. *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal regression, and Survival Analysis*. Spinger, 2015.

F. E. Harrell, R. M. Califf, D. B. Pryor, K. L. Lee, and R. A. Rosati. Evaluating the yield of medical tests. *Journal of the American Medical Association*, 247(18):2543–2546, 1982.

F. E. Harrell Jr, K. L. Lee, and D. B. Mark. Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors. *Statistics in Medicine*, 15(4):361–387, 1996.

T. Hashimoto, M. Srivastava, H. Namkoong, and P. Liang. Fairness without demographics in repeated loss minimization. In *International Conference on Machine Learning*, pages 1929–1938. PMLR, 2018.

A. G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58 (1):83–90, 1971.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

S. Hu and G. H. Chen. Fairness in survival analysis with distributionally robust optimization. *Journal of Machine Learning Research*, 25(246):1–85, 2024.

D. Hubbard, B. Rostykus, Y. Raimond, and T. Jebara. Beta survival models. In *Survival Prediction - Algorithms, Challenges and Applications*, pages 22–39. PMLR, 2021.

W. T. Huh, R. Levi, P. Rusmevichientong, and J. B. Orlin. Adaptive data-driven inventory control with censored demand based on Kaplan-Meier estimator. *Operations Research*, 59(4):929–941, 2011.

H. Hung and C.-T. Chiang. Estimation methods for time-dependent auc models with survival data. *Canadian Journal of Statistics*, 38(1):8–26, 2010.

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

V. Jeanselme, B. Tom, and J. Barrett. Neural survival clustering: Non-parametric mixture of neural networks for survival clustering. In *Conference on Health, Inference, and Learning*, pages 92–102. PMLR, 2022.

V. Jeanselme, C. H. Yoon, B. Tom, and J. Barrett. Neural fine-gray: Monotonic neural networks for competing risks. In *Conference on Health, Inference, and Learning*, pages 379–392. PMLR, 2023.

V. Jeanselme, N. Agarwal, and C. Wang. Review of language models for survival analysis. In *AAAI 2024 Spring Symposium on Clinical Foundation Models*, 2024.

J. D. Kalbfleisch and R. L. Prentice. *The Statistical Analysis of Failure Time Data*. John Wiley & Sons, 1980.

E. L. Kaplan and P. Meier. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53(282):457–481, 1958.

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

K. N. Keya, R. Islam, S. Pan, I. Stockwell, and J. Foulds. Equitable allocation of healthcare resources with fair survival models. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, pages 190–198. SIAM, 2021.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2015.

J. P. Klein and M. L. Moeschberger. *Survival Analysis: Techniques for Censored and Truncated Data, Second Edition*. Springer, 2003.

J. P. Klein, H. C. Van Houwelingen, J. G. Ibrahim, and T. H. Scheike. *Handbook of Survival Analysis*. CRC Press, 2016.

D. G. Kleinbaum and M. Klein. *Survival Analysis: A Self-Learning Text, Third Edition*. Springer, 2012.

W. A. Knaus, F. E. Harrell, J. Lynn, L. Goldman, R. S. Phillips, A. F. Connors, N. V. Dawson, W. J. Fulkerson, R. M. Califf, N. Desbiens, P. Layde, R. K. Oye, P. E. Bellamy, R. B. Hakim, and D. P. Wagner. The SUPPORT prognostic model: Objective estimates of survival for seriously ill hospitalized adults. *Annals of Internal Medicine*, 122(3):191–203, 1995.

M. S. Kovalev, L. V. Utkin, and E. M. Kasimov. SurvLIME: A method for explaining machine learning survival models. *Knowledge-Based Systems*, 2020.

J. A. Koziol and Z. Jia. The concordance index C and the Mann–Whitney parameter pr($x > y$) with randomly censored data. *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, 51(3): 467–474, 2009.

S. Kpotufe and N. Verma. Time-accuracy tradeoffs in kernel prediction: controlling prediction quality. *Journal of Machine Learning Research*, 2017.

M. Krzyziński, M. Spytek, H. Baniecki, and P. Biecek. SurvSHAP(t): Time-dependent explanations of machine learning survival models. *Knowledge-Based Systems*, 2023.

H. Kvamme and Ø. Borgan. Continuous and discrete-time survival prediction with neural networks. *Lifetime Data Analysis*, 27(4):710–736, 2021.

H. Kvamme, Ø. Borgan, and I. Scheel. Time-to-event prediction with neural networks and Cox regression. *Journal of Machine Learning Research*, 20(129):1–30, 2019.

J. Lambert and S. Chevret. Summary measure of discrimination in survival models based on cumulative/dynamic time-dependent roc curves. *Statistical Methods in Medical Research*, 25(5):2088–2102, 2016.

C. Lee, W. Zame, J. Yoon, and M. Van Der Schaar. Deephit: A deep learning approach to survival analysis with competing risks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

C. Lee, J. Yoon, and M. Van Der Schaar. Dynamic-DeepHit: A deep learning approach for dynamic survival analysis with competing risks based on longitudinal data. *IEEE Transactions on Biomedical Engineering*, 67(1):122–133, 2019.

J. Lei, A. Rinaldo, and L. Wasserman. A conformal prediction approach to explore functional data. *Annals of Mathematics and Artificial Intelligence*, 74(1-2):29–43, 2015.

J. Lei, M. G'Sell, A. Rinaldo, R. J. Tibshirani, and L. Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113(523):1094–1111, 2018.

J. Li and S. Ma. *Survival Analysis in Medicine and Genetics*. CRC Press, 2013.

M. Li, H. Namkoong, and S. Xia. Evaluating model performance under worst-case subpopulations. In *Advances in Neural Information Processing Systems*, 2021.

W. Liu, R. Lin, Z. Liu, L. Xiong, B. Schölkopf, and A. Weller. Learning with hyperspherical uniformity. In *International Conference On Artificial Intelligence and Statistics*, pages 1180–1188. PMLR, 2021.

S. M. Lundberg and S.-I. Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, 2017.

D. Machin, Y. B. Cheung, and M. Parmar. *Survival Analysis: A Practical Approach*. John Wiley & Sons, 2006.

L. Manduchi, R. Marcinkevičs, M. C. Massi, T. Weikert, A. Sauter, V. Gotta, T. Müller, F. Vasella, M. C. Neidert, M. Pfister, B. Stieltjes, and J. E. Vogt. A deep variational approach to clustering survival data. In *International Conference on Learning Representations*, 2022.

N. R. Mann, R. E. Schafer, and N. D. Singpurwalla. *Methods for Statistical Analysis of Reliability and Life Data*. John Wiley & Sons, 1974.

N. Mantel. Evaluation of survival data and two new rank order statistics arising in its consideration. *Cancer Chemotherapy Reports*, 50(3):163–170, 1966.

C. Molnar. *Interpretable Machine Learning*. 2 edition, 2022. URL `https://christophm.github.io/interpretable-ml-book`.

M. Monod, P. Krusche, Q. Cao, B. Sahiner, N. Petrick, D. Ohlssen, and T. Coroller. Torchsurv: A lightweight package for deep survival analysis, 2024.

I. Moon, S. Groha, and A. Gusev. SurvLatent ODE: A neural ode based time-to-event model with competing risks for longitudinal data improves cancer-associated venous thromboembolism (vte) prediction. In *Machine Learning for Healthcare Conference*, 2022.

C. Nagpal, X. Li, and A. Dubrawski. Deep survival machines: Fully parametric survival regression and representation learning for censored data with competing risks. *IEEE Journal of Biomedical and Health Informatics*, 25(8):3163–3175, 2021a.

C. Nagpal, S. Yadlowsky, N. Rostamzadeh, and K. Heller. Deep Cox mixtures for survival regression. In *Machine Learning for Healthcare Conference*, pages 674–708. PMLR, 2021b.

C. Nagpal, M. Goswami, K. Dufendach, and A. Dubrawski. Counterfactual phenotyping with censored time-to-events. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3634–3644, 2022a.

C. Nagpal, W. Potosnak, and A. Dubrawski. auton-survival: An open-source package for regression, counterfactual estimation, evaluation and phenotyping with censored time-to-event data. In *Machine Learning for Healthcare Conference*, pages 585–608. PMLR, 2022b.

K. Namboodiri and C. M. Suchindran. *Life Table Techniques and Their Applications*. Academic Press, 2013.

W. Nelson. Hazard plotting for incomplete failure data. *Journal of Quality Technology*, 1:27–52, 1969.

H. Papadopoulos, K. Proedrou, V. Vovk, and A. Gammerman. Inductive confidence machines for regression. In *European Conference on Machine Learning*, pages 345–356. Springer, 2002.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Y. Peng and B. Yu. *Cure Models: Methods, Applications, and Implementation*. CRC Press, 2021.

S. Pölsterl. scikit-survival: A library for time-to-event analysis built on top of scikit-learn. *Journal of Machine Learning Research*, 21(1):8747–8752, 2020.

R. L. Prentice and J. D. Kalbfleisch. Hazard rate models with covariates. *Biometrics*, pages 25–39, 1979.

R. L. Prentice and S. Zhao. *The Statistical Analysis of Multivariate Failure Time Data: A Marginal Modeling Approach*. CRC Press, 2019.

P. Putzel, H. Do, A. Boyd, H. Zhong, and P. Smyth. Dynamic survival analysis for EHR data with personalized parametric distributions. In *Machine Learning for Healthcare Conference*, pages 648–673. PMLR, 2021.

S.-A. Qi, N. Kumar, M. Farrokh, W. Sun, L.-H. Kuan, R. Ranganath, R. Henao, and R. Greiner. An effective meaningful way to evaluate survival models. In *International Conference on Machine Learning*, volume 202, pages 28244–28276. PMLR, 2023.

S.-a. Qi, W. Sun, and R. Greiner. SurvivalEVAL: A comprehensive open-source python package for evaluating individual survival distributions. In *Proceedings of the 2023 AAAI Fall Symposia*, 2024a.

S.-a. Qi, Y. Yu, and R. Greiner. Conformalized survival distributions: A generic post-process to increase calibration. In *International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 41303–41339. PMLR, 2024b.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021. URL `https://www.R-project.org/`.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

M. M. Rahman and S. Purushotham. Fair and interpretable models for survival analysis. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1452–1462, 2022.

V. C. Raykar, H. Steck, B. Krishnapuram, C. Dehing-oberije, and P. Lambin. On ranking in survival analysis: Bounds on the concordance index. In *Advances in Neural Information Processing Systems*, 2007.

M. T. Ribeiro, S. Singh, and C. Guestrin. "why should I trust you?" Explaining the predictions of any classifier. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144, 2016.

D. Rindt, R. Hu, D. Steinsaltz, and D. Sejdinovic. Survival regression with proper scoring rules and monotonic neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 1190–1205. PMLR, 2022.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 1959.

M. Schumacher, G. Bastert, H. Bojar, K. Huebner, M. Olschewski, W. Sauerbrei, C. Schmoor, C. Beyerle, R. L. Neumann, and H. F. Rauschecker. Randomized 2 x 2 trial evaluating hormonal treatment and the duration of chemotherapy in node-positive breast cancer patients. german breast cancer study group. *Journal of Clinical Oncology*, 12(10):2086–2093, 1994.

S. Selvin. *Survival Analysis for Epidemiologic and Medical Research*. Cambridge University Press, 2008.

O. Shchur, M. Biloš, and S. Günnemann. Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*, 2020.

X. Shen, J. Elmer, and G. H. Chen. Neurological prognostication of post-cardiac-arrest coma patients using eeg data: A dynamic survival analysis framework with competing risks. In *Machine Learning for Healthcare Conference*, pages 667–690. PMLR, 2023.

N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for cox's proportional hazards model via coordinate descent. *Journal of Statistical Software*, 39(5):1, 2011.

E. Steinberg, J. A. Fries, Y. Xu, and N. Shah. MOTOR: A time-to-event foundation model for structured medical records. In *International Conference on Learning Representations*, 2024.

X. Sun and P. Qiu. NSOTree: Neural survival oblique tree. *arXiv preprint arXiv:2309.13825*, 2023.

W. Tang, K. He, G. Xu, and J. Zhu. Survival analysis via ordinary differential equations. *Journal of the American Statistical Association*, 2022a.

W. Tang, J. Ma, Q. Mei, and J. Zhu. SODEN: A scalable continuous-time survival model through ordinary differential equation networks. *Journal of Machine Learning Research*, 23(34):1–29, 2022b.

R. Tibshirani. The lasso method for variable selection in the cox model. *Statistics in Medicine*, 16(4): 385–395, 1997.

G. Tutz and M. Schmid. *Modeling Discrete Time-to-Event Data*. Springer, 2016.

H. Uno, T. Cai, L. Tian, and L.-J. Wei. Evaluating prediction rules for t-year survivors with censored regression models. *Journal of the American Statistical Association*, 102(478):527–537, 2007.

H. Uno, T. Cai, M. J. Pencina, R. B. D'Agostino, and L.-J. Wei. On the c-statistics for evaluating overall adequacy of risk prediction procedures with censored survival data. *Statistics in Medicine*, 30(10):1105–1117, 2011.

L. Van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008.

R. Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge University Press, 2018.

P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

V. Vovk, A. Gammerman, and G. Shafer. *Algorithmic Learning in a Random World*. Springer Science & Business Media, 2005.

P. Wang, Y. Li, and C. K. Reddy. Machine learning for survival analysis: A survey. *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019.

T. Wang and P. Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning*, 2020.

S. Wiegrebe, P. Kopper, R. Sonabend, and A. Bender. Deep learning for survival analysis: A review. *arXiv preprint arXiv:2305.14961*, 2023.

J. Xu and Y. Peng. Nonparametric cure rate estimation with covariates. *Canadian Journal of Statistics*, 42(1):1–17, 2014.

Y. Xu, N. Ignatiadis, E. Sverdrup, S. Fleming, S. Wager, and N. Shah. Treatment heterogeneity with survival outcomes. In *Handbook of Matching and Weighting Adjustments for Causal Inference*, pages 445–482. Chapman and Hall/CRC, 2023.

H. Yanagisawa, K. Miyaguchi, and T. Katsuki. Hierarchical lattice layer for partially monotone neural networks. In *Advances in Neural Information Processing Systems*, 2022.

C.-N. Yu, R. Greiner, H.-C. Lin, and V. Baracos. Learning patient-specific cancer survival distributions as a sequence of dependent regressors. In *Advances in Neural Information Processing Systems*, 2011.

A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. `https://D2L.ai`.

Q. Zhang, A. Lipani, O. Kirnap, and E. Yilmaz. Self-attentive Hawkes process. In *International Conference on Machine Learning*, pages 11183–11193. PMLR, 2020.

R. Zhang, R. Xin, M. Seltzer, and C. Rudin. Optimal sparse survival trees. In *International Conference on Artificial Intelligence and Statistics*, pages 352–360. PMLR, 2024.

W. Zhang and J. C. Weiss. Longitudinal fairness with censorship. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

Q. Zhong, J. W. Mueller, and J.-L. Wang. Deep extended hazard models for survival analysis. In *Advances in Neural Information Processing Systems*, 2021.

Q. Zhong, J. Mueller, and J.-L. Wang. Deep learning for the partially linear Cox model. *The Annals of Statistics*, 50(3):1348–1375, 2022.

H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B*, 67(2):301–320, 2005.

S. Zuo, H. Jiang, Z. Li, T. Zhao, and H. Zha. Transformer Hawkes process. In *International Conference on Machine Learning*, pages 11692–11702. PMLR, 2020.

B. Zupan, J. Demšar, M. W. Kattan, J. R. Beck, and I. Bratko. Machine learning for survival analysis: a case study on recurrence of prostate cancer. *Artificial Intelligence in Medicine*, 20(1):59–75, 2000.