Chapter 1

Introduction

Writing in the middle of a global pandemic, applications of survival analysis are more relevant than ever. Predicting the time from onset of COVID-19 symptoms to hospitalisation, or the time from hospitalisation to intubation, or intubation to death, are all time-to-event predictions that are at the centre of survival analysis. As well as morbid applications, survival analysis predictions may be concerned with predicting the time until a customer cancels their gym membership, or the lifetime of a lightbulb; any event that is guaranteed (or at least very likely) to occur can be modelled by a survival analysis prediction. As these predictions can be so sensitive, for example a model predicting when a child should be taken off breathing support [62], the best possible predictions, evaluated to the highest standard, are a necessity. In other fields of predictive modelling, machine learning has made incredible breakthroughs (such as AlphaFold[1]), therefore applying machine learning to survival analysis is a natural step in the evolution of an important field.

Survival analysis is the field of Statistics focusing on modelling the distribution of an event, which may mean the time until the event takes place, the risk of the event happening, the probability of the event occurring at a single time, or the event's underlying probability distribution. Survival analysis ('survival') is a unique field of study in Statistics as it includes the added difficulty of 'censoring'. Censoring is best described through example: a study is conducted to determine the mortality rate of a group of patients after diagnoses with a particular disease. If a patient dies during this study then their outcome is 'death' and their time of death can be recorded. However if a patient drops-out of the study before they die, then their time of death (though guaranteed to occur) is unknown and the only available information is the time at which they left the study. This patient is now said to be *censored* at the time they drop out. The censoring mechanism allows as much outcome information (time and event) to be captured as possible for all patients (observations).

Machine learning (ML) is the field of Statistics primarily concerned with building models to either predict outputs from inputs or to learn relationships from data [118, 145]. This thesis is limited to the former case, or more specifically supervised learning, as this is the field in which the vast majority of survival

[1] `https://deepmind.com/research/case-studies/alphafold`

problems live. Relative to other areas of supervised learning, development in survival analysis has been slow – the majority of developments in machine learning for survival analysis have only been in the past decade (see chapters 3-4). This appears to have resulted in less interest in the development of machine learning survival models (chapter 3), less rigour in the evaluation of such models (chapter 4), and fewer off-shelf/open-source implementations (chapter 6).

2.4.2. Machine Learning in Classification and Regression

Before introducing machine learning for survival analysis, which is considered 'non-classical', the more standard classification and regression set-ups are provided; these are referenced throughout this thesis.

2.4.2.1. Classification

Classification problems make predictions about categorical (or discrete) events, these may be *deterministic* or *probabilistic*. Deterministic classification predicts which category an observation falls into, whereas probabilistic classification predicts the probability of an observation falling into each category. In this brief introduction only binary single-label classification is discussed, though the multi-label case is considered in section 5.5.7.4. In binary classification, there are two possible categories an observation can fall into, usually referred to as the 'positive' and 'negative' class. For example predicting the probability of death due to a virus is a probabilistic classification task where the 'positive' event is death.

A probabilistic prediction is more informative than a deterministic one as it encodes uncertainty about the prediction. For example it is clearly more informative to predict a 70% chance of rain tomorrow instead of simply 'rain'. Moreover the latter prediction implicitly contains an erroneous assumption of certainty, e.g. 'it will rain tomorrow'.

Box 1 (Classification Task)**.** Let (X, Y) be random variables t.v.i. $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{X} \subseteq \mathbb{R}^p$ and $\mathcal{Y} = \{0, 1\}$. Then,

i) The *probabilistic classification task* is the problem of predicting the probability of a single event taking place and is specified by $g : \mathcal{X} \to [0, 1]$.

ii) The *deterministic classification task* is the problem of predicting if a single event takes place and is specified by $g : \mathcal{X} \to \mathcal{Y}$.

The estimated prediction functional $\hat{g}$ is fit on training data $(X_1, Y_1), ..., (X_n, Y_n) \overset{i.i.d.}{\sim} (X, Y)$ and is considered 'good' if $\mathbb{E}[L(Y^*, \hat{g}(X^*))]$ is low, where $(X^*, Y^*) \sim (X, Y)$ is independent of $(X_1, Y_1), ..., (X_n, Y_n)$ and $\hat{g}$.

In the probabilistic case, the prediction $\hat{g}$ maps to the estimated probability mass function $\hat{p}_Y$ s.t. $\hat{p}_Y(1) = 1 - \hat{p}_Y(0)$.

2.4.2.2. Regression

A regression prediction is one in which the goal is to predict a continuous outcome from a set of features. For example predicting the time until an event (without censoring) occurs, is a regression problem.

Box 2 (Regression Task)**.** Let (X, Y) be random variables t.v.i. $\mathcal{X} \times \mathcal{Y}$ where $\mathcal{X} \subseteq \mathbb{R}^p$ and $\mathcal{Y} \subseteq \mathbb{R}$. Let $\mathcal{S} \subset \text{Distr}(\mathcal{Y})$ be a convex set of distributions on $\mathcal{Y}$. Then,

i) The *probabilistic regression task* is the problem of predicting a conditional distribution over the Reals and is specified by $g : \mathcal{X} \to \mathcal{S}$.

ii) The *deterministic regression task* is the problem of predicting a single continuous value in the Reals and is specified by $g : \mathcal{X} \to \mathcal{Y}$.

The estimated prediction functional $\hat{g}$ is fit on training data $(X_1, Y_1), ..., (X_n, Y_n) \overset{i.i.d.}{\sim} (X, Y)$ and is considered 'good' if $\mathbb{E}[L(Y^*, \hat{g}(X^*))]$ is low, where $(X^*, Y^*) \sim (X, Y)$ is independent of $(X_1, Y_1), ..., (X_n, Y_n)$ and $\hat{g}$.

Whilst regression can be either probabilistic or deterministic, the latter is much more common and therefore in this thesis 'regression' refers to the deterministic case unless otherwise stated.

2.5. Survival Analysis Task

The survival prediction problems identified in section 2.3 are now formalised as machine learning tasks.

Box 3 (Survival Task)**.** Let (X, T, Δ) be random variables t.v.i. $\mathcal{X} \times \mathcal{T} \times \{0, 1\}$ where $\mathcal{X} \subseteq \mathbb{R}^p$ and $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}$. Let $\mathcal{S} \subseteq \text{Distr}(\mathcal{T})$ be a convex set of distributions on $\mathcal{T}$ and let $\mathcal{R} \subseteq \mathbb{R}$. Then,

i) The *probabilistic survival task* is the problem of predicting a conditional distribution over the positive Reals and is specified by $g : \mathcal{X} \to \mathcal{S}$.

ii) The *deterministic survival task* is the problem of predicting a continuous value in the positive Reals and is specified by $g : \mathcal{X} \to \mathcal{T}$.

iii) The *survival ranking task* is specified by predicting a continuous ranking in the Reals and is specified by $g : \mathcal{X} \to \mathcal{R}$.

The estimated prediction functional $\hat{g}$ is fit on training data $(X_1, T_1, \Delta_1), ..., (X_n, T_n, \Delta_n) \overset{i.i.d.}{\sim} (X, T, \Delta)$ and is considered 'good' if $\mathbb{E}[L(T^*, \Delta^*, \hat{g}(X^*))]$ is low, where $(X^*, T^*, \Delta^*) \sim (X, T, \Delta)$ is independent of $(X_1, T_1, \Delta_1), ..., (X_n, T_n, \Delta_n)$ and $\hat{g}$.

3.5. Support Vector Machines

3.5.1. SVMs for Regression

In the simplest explanation, support vector machines (SVMs) [58] fit a hyperplane, g, on given training data and make predictions for new values as $\hat{g}(X^*)$ for some testing covariate X^*. One may expect the hyperplane to be fit so that all training covariates would map perfectly to the observed labels (a 'hard-boundary') however this would result in overfitting and instead an acceptable ('soft'-)boundary of error, the 'ϵ-tube', dictates how 'incorrect' predictions may be, i.e. how large an underestimate or overestimate. Figure 6 visualises support vector machines for regression with a linear hyperplane g, and an acceptable boundary of error within the dashed lines (the ϵ-tube). SVMs are not limited to linear boundaries and *kernel* functions are utilised to specify more complex hyperplanes. Exact details of the optimization/separating procedure are not discussed here but many off-shelf 'solvers' exist in different programming languages for fitting SVMs.

In the regression setting, the goal of SVMs is to estimate the function

$$g : \mathbb{R}^p \to \mathbb{R}; \quad (x) \mapsto x\beta + \beta_0 \tag{3.5.1}$$

by estimation of the weights $\beta \in \mathbb{R}^p, \beta_0 \in \mathbb{R}$ via the optimisation problem

$$\begin{aligned} &\min_{\beta,\beta_0,\xi,\xi^*} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ &\text{s.t.} \begin{cases} Y_i - g(X_i) & \leq \epsilon + \xi_i \\ g(X_i) - Y_i & \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* & \geq 0, \ i = 1, ..., n \end{cases} \end{aligned} \tag{3.5.2}$$

where $C \in \mathbb{R}$ is the regularization/cost parameter, ξ_i, ξ_i^* are slack parameters and ϵ is a margin of error for observations on the wrong side of the hyperplane, and g is defined in eq. (3.5.1). The effect of the slack parameters is seen in fig. 6 in which a maximal distance from the ϵ-tube is dictated by the slack variables.

In fitting, the dual of the optimisation is instead solved and substituting the optimised parameters into eq. (3.5.1) gives the prediction function,

$$\hat{g}(X^*) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)K(X^*, X_i) + \beta_0 \tag{3.5.3}$$

where α_i, α_i^* are Lagrangrian multipliers and K is some kernel function.[1] The Karush-Kuhn-Tucker conditions required to solve the optimisation for α result in the key property of SVMs, which is that values $\alpha_i = \alpha_i^* = 0$ indicate that observation i is 'inside' the ϵ-tube and if $\alpha_i \neq 0$ or $\alpha_i^* \neq 0$ then i is outside the

[1] Discussion about the purpose of kernels and sensible choices can be found in [84, 145, 311].

tube and termed a *support vector*. It is these 'support vectors' that influence the shape of the separating boundary.

The choice of kernel and its parameters, the regularization parameter C, and the acceptable error ϵ, are all tunable hyper-parameters, which makes the support vector machine a highly adaptable and often well-performing machine learning method. However the parameters C and ϵ often have no clear apriori meaning (especially true when predicting abstract rankings) and thus require extensive tuning over a great range of values; no tuning will result in a very poor model fit.

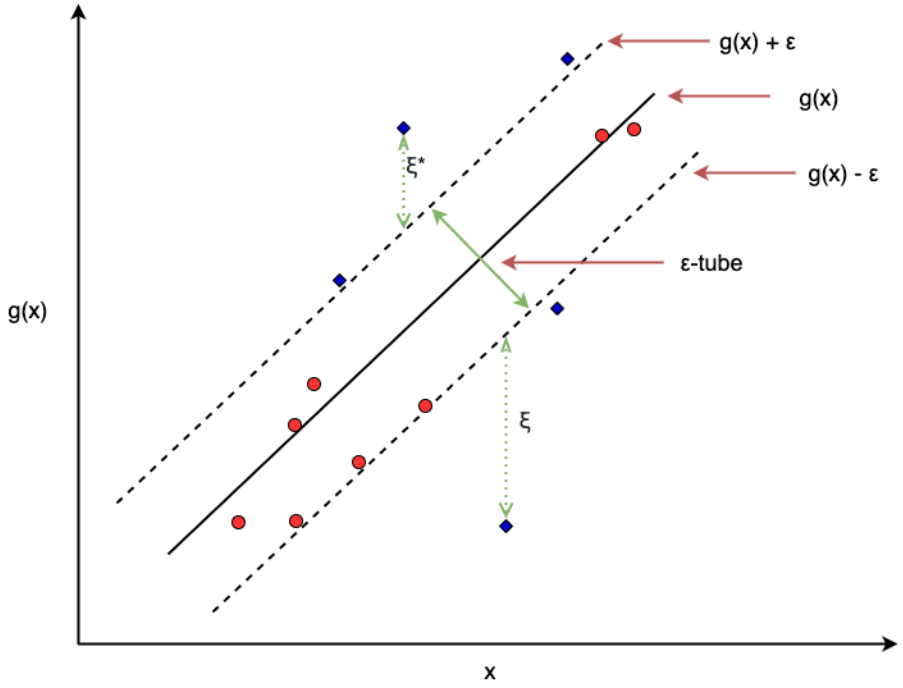


Figure 6: Visualising a support vector machine with an ϵ-tube and slack parameters ξ and ξ^*. Red circles are values within the ϵ-tube and blue diamonds are values outside the tube. x-axis is single covariate, x, and y-axis is $g(x) = x\beta + \beta_0$.

3.5.2. SVMs for Survival Analysis

Similarly to random forests, all research for Survival Support Vector Machines (SSVMs) can be reduced to very few algorithms, in fact only one unique off-shelf algorithm is identified in this survey. No SSVM for distribution predictions exist, instead they either predict survival time, rankings, or a hybrid of the two.

Other reviews and surveys of SSVMs include a short review by Wang *et al.* (2017) [317] and some benchmark experiments and short surveys from Van Belle *et al.* (2011) [306], Goli *et al.* (2016) [102] and Fouodo *et al.* (2018) [84]. All the benchmark experiments in these papers indicate that the Cox PH performs as well as, if not better than, the SSVMs.

Initial attempts at developing SSVMs by Shivaswamy *et al.* (2007) [273] took the most 'natural' course and attempt to treat the problem as a regression one

with adjustments in the optimisation for censoring. These methods have a natural interpretation and are intuitive in their construction. Further development of these by Khan and Zubek (2008) [158] and Land *et al.* (2011) [177] focused on different adjustments for censoring in order to best reflect a realistic survival data set-up. Simultaneously, ranking models were developed in order to directly optimise a model's discriminatory power. Developments started with the work of Evers and Messow (2008) [76] but were primarily made by Van Belle *et al.* (2007)-(2011) [302, 303, 304, 305]. These lack the survival time interpretation but are less restrictive in the optimisation constraints. Finally a hybrid of the two followed naturally from Van Belle *et al.* (2011) [306] by combining the constraints from both the regression and ranking tasks. This hybrid method allows a survival time interpretation whilst still optimising discrimination. These hybrid models have become increasingly popular in not only SSVMs, but also neural networks (section 3.6). Instead of presenting these models chronologically, the final hybrid model is defined and then other developments can be more simply presented as components of this hybrid. One model with an entirely different formulation is considered after the hybrid.

For all SSVMs defined in this section let: ξ_i, ξ_i^*, ξ_i' be slack variables; β, β_0 be model weights in $\mathbb{R}$; C, μ be regularisation hyper-parameters in $\mathbb{R}$; $(X_i, T_i, \Delta_i) \overset{i.i.d.}{\sim} (X, T, \Delta)$ be the usual training data; and $g(x) = x\beta + \beta_0$.

SSVM-Hybrid Van Belle *et al.* published several papers developing SSVMs, which culminate in the hybrid model here termed 'SSVM-Hybrid' [306]. The model is defined by the optimisation problem,

SSVM-Hybrid

$$\begin{aligned} &\min_{\beta,\beta_0,\xi,\xi',\xi^*} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}\xi_i + \mu\sum_{i=1}^{n}(\xi_i' + \xi_i^*) \\ &\text{s.t.} \begin{cases} g(X_i) - g(X_{j(i)}) \geq T_i - T_{j(i)} - \xi_i, \\ \Delta_i(g(X_i) - T_i) \leq \xi_i^* \\ T_i - g(X_i) \leq \xi_i' \\ \xi_i, \xi_i', \xi_i^* \geq 0, \quad \forall i = 1, ..., n \end{cases} \end{aligned} \tag{3.5.4}$$

where $j(i) := \text{argmax}_{j \in 1,...n}\{T_j : T_j < T_i\}$ is an index discussed further below. A prediction for test data is given by,

$$\hat{g}(X^*) = \sum_{i=1}^{n} \alpha_i(K(X_i, X^*) - K(X_{j(i)}, X^*)) + \alpha_i^* K(X_i, X^*) - \Delta_i \alpha_i' K(X_i, X^*) + \beta_0 \tag{3.5.5}$$

where $\alpha_i, \alpha_i^*, \alpha_i'$ are Lagrange multipliers and K is a chosen kernel function, which may have hyper-parameters to select or tune.

SVCR (Regression) Examining the components of the SSVM-Hybrid model will help identify its relation to previously published SSVMs. First note the

model's connection to the regression setting when on setting $C = 0$, removing the associated first constraint and ignoring Δ in the second constraint, the regression setting is exactly recovered:

$$\min_{\beta,\beta_0,\xi,\xi'} \frac{1}{2}\|\beta\|^2 + \mu\sum_{i=1}^{n}(\xi_i + \xi_i') \\ \text{s.t.} \begin{cases} g(X_i) - T_i \leq \xi_i \\ T_i - g(X_i) \leq \xi_i' \\ \xi_i, \xi_i' \geq 0, \quad \forall i = 1, ..., n \end{cases} \tag{3.5.6}$$

Note a slight difference in the formulation of this optimisation to the original regression problem, here no error component ϵ is directly included, instead this is part of the optimisation and considered as part of the slack parameters ξ_i, ξ_i'; effectively this is the same as setting $\epsilon = 0$. This formulation removes the ϵ-tube symmetry seen previously and therefore distinguishes more clearly between over-estimates and underestimates, with each being penalised differently. Removing the ϵ parameter can lead to model overfitting as all points become support vectors, however careful tuning of other hyper-parameters can effectively control for this.

This formulation allows for clearer control over left-, right-, and un-censored observations. Clearly if an observation is uncensored then the true value is known and should be predicted exactly, hence under- and over-estimates are equally problematic and should be penalised the same. If an observation is right-censored then the true death time is greater than the observed time and therefore overestimates should not be heavily penalised but underestimates should be; conversely for left-censored observations.

This leads to the first SSVM for regression from Shivaswamy *et al.* (2007) [273].

SVCR

$$\min_{\beta,\beta_0,\xi,\xi^*} \frac{1}{2}\|\beta\|^2 + \mu\Big(\sum_{i\in R}\xi_i + \sum_{i\in L}\xi_i^*\Big) \\ \text{s.t.} \begin{cases} g(X_i) - T_i \leq \xi_i^*, \quad \forall i \in R \\ T_i - g(X_i) \leq \xi_i, \quad \forall i \in L \\ \xi_i \geq 0, \forall i \in R; \xi_i^* \geq 0, \forall i \in L \end{cases} \tag{3.5.7}$$

where L is the set of observations who are either left- or un-censored, and R is the set of observations who are either right- or un-censored. Hence an uncensored observation is constrained on both sides as their true survival time is known, whereas a left-censored observation is constrained in the amount of 'over-prediction' and a right-censored observation is constrained by 'under-prediction'. This is intuitive as the only known for these censoring types are the lower and upper bounds of the actual survival time respectively.

Reducing this to the thesis scope of right-censoring only results in the opti-

misation:

$$\min_{\beta,\beta_0,\xi,\xi^*} \frac{1}{2}\|\beta\|^2 + \mu\Big(\sum_{i=1}^{n} \xi_i + \xi_i^*\Big)$$
$$\text{s.t.} \begin{cases} \Delta_i(g(X_i) - T_i) \leq \xi_i \\ T_i - g(X_i) \leq \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \\ \forall i \in 1, ..., n \end{cases} \tag{3.5.8}$$

which can be seen to be identical to SSVM-Hybrid when $C = 0$ and the first constraint is removed. Predictions are found by,

$$\hat{g}(X^*) = \sum_{i=1}^{n} \alpha_i^* K(X_i, X^*) - \Delta_i \alpha_i' K(X_i, X^*) + \beta_0 \tag{3.5.9}$$

The advantage of this algorithm is its simplicity. Clearly if no-one is censored then the optimisation is identical to the regression optimisation in eq. (3.5.2). As there is no ϵ hyper-parameter, the run-time complexity is the same as, if not quicker than, a regression SVM. Both left- and right-censoring are handled and no assumptions are made about independent censoring. With respect to performance, benchmark experiments [84] indicate that the SVCR does not outperform a naïve SVR (i.e. censoring ignored). The SVCR is implemented in the R package **survivalsvm** [84] and is referred to as 'regression'.

As discussed, the error margin for left- and right- censoring should not necessarily be equal and the penalty for each should not necessarily be equal either. Hence a natural extension to SVCR is to add further parameters to better separate the different censoring types, which gives rise to the SVRc [158]. However this model is only briefly discussed as left-censoring is out of scope of this thesis and also the model is patented and therefore not easily accessible. The model is given by the optimisation,

SVRc

$$\min_{\beta,\beta_0,\xi,\xi^*} \frac{1}{2}\|\beta\|^2 + \sum_{i=1}^{n} C_i\xi_i + C_i^*\xi_i'$$
$$\text{s.t.} \begin{cases} g(X_i) - T_i \leq \epsilon_i' + \xi_i' \\ T_i - g(X_i) \leq \epsilon_i + \xi_i \\ \xi_i, \xi_i' \geq 0, \quad \forall i = 1, ..., n \end{cases} \tag{3.5.10}$$

Where $C_i = \Delta_i C_c + (1 - \Delta_i)C_n, \epsilon_i = \Delta_i\epsilon_c + (1 - \Delta_i)\epsilon_n$ and analogously for $C_i^*, C_C^*, \epsilon^*, ...$ The new hyper-parameters $C_c, C_n, \epsilon_c, \epsilon_n$ are the penalty for errors in censored predictions (c) and uncensored predictions (n) for left and right (*) censoring, and the acceptable margin of errors respectively. The rationale behind this algorithm is clear, by having asymmetric error margins the algorithm can

penalise predictions that are clearly wrong whilst allowing predictions that may be correct (but ultimately unknown due to censoring). Experiments indicate the model may have superior discrimination than the Cox PH [158] and SVCR [72]. However these conclusions are weak as independent experiments do not have access to the patented model.

The largest drawback of the algorithm is a need to tune eight parameters. As the number of hyper-parameters to tune increases, so too does model fitting time as well as the risk of overfitting. The problem of extra hyper-parameters is the most common disadvantage of the model given in the literature [84, 177]. Land *et al.* (2011) [177] present an adaptation to the SVRc to improve model fitting time, termed the EP-SVRc, which uses Evolutionary Programming to determine the optimal values for the parameters. No specific model or algorithm is described, nor any quantitative results presented. No evidence can be found for this method being used since publication. The number of hyper-parameters in the SVRc, coupled with its lack of accessibility, outweigh the benefits of the claimed predictive performance and is therefore clearly not APT and will not be considered further.

SSVM-Rank The regression components of SSVM-Hybrid (eq. (3.5.4)) have been fully examined, now turning to the ranking components and setting $\mu = 0$. In this case the model reduces to

SSVM-Rank

$$\begin{aligned} &\min_{\beta,\beta_0,\xi} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}\xi_i \\ &\text{s.t.} \begin{cases} g(X_i) - g(X_{j(i)}) \geq T_i - T_{j(i)} - \xi_i, \\ \xi_i \geq 0, \quad \forall i = 1, ..., n \end{cases} \end{aligned} \tag{3.5.11}$$

with predictions

$$\hat{g}(X^*) = \sum_{i=1}^{n} \alpha_i (K(X_i, X^*) - K(X_{j(i)}, X^*)) \tag{3.5.12}$$

This formulation, termed here 'SSVM-Rank', has been considered by numerous authors in different forms, including Evers and Messow [76] and Van Belle *et al.* [303, 304, 306]. The primary differences between the various models are in which observations are compared in order to optimise discrimination; to motivate why this matters, first observe the intuitive nature of the optimisation constraints. By example, define $k := T_i - T_{j(i)}$ and say $T_i > T_{j(i)}$. Then, in the first constraint, $g(X_i) - g(X_{j(i)}) \geq k - \xi_i$. As $k > 0$ and $\xi_i \geq 0$, it follows that $g(X_i) > g(X_{j(i)})$, hence creating a concordant ranking[1] which is the opposite to the between observations i (ranked higher) and $j(i)$; illustrating why this optimisation results in a ranking model.

This choice of comparing observations i and $j(i)$ (defined below) stems from a few years of research in an attempt to optimise the algorithm with respect to both

[1]Note this ranking has the interpretation 'higher rank equals lower risk'.

speed and predictive performance. In the original formulation, RANKSVMC [303], the model ranks all possible pairs of observations. This is clearly infeasible as it increases the problem to a $\mathcal{O}(qn^2/2)$ runtime where q is the proportion of non-censored observations out of a total sample size n [304]. The problem was reduced by taking a nearest neighbours approach and only considering the kth closest observations [304]. Simulation experiments determined that the single nearest neighbour was sufficient, thus arriving at $j(i)$, the observation with the largest observed survival time smaller than T_i,

$$j(i) := \underset{j \in 1,...n}{\text{argmax}} \{T_j : T_j < T_i\} \tag{3.5.13}$$

This requires that the first observation is taken to be an event, even if it is actually censored. In practice, sorting observations by survival time then greatly speeds up the model run-time [84]. The RANKSVMC and SSVM-RANK are implemented in **survivalsvm** [84] and referred to as 'vanbelle1' and 'vanbelle2' respectively.

The hybrid model is repeated below with the ranking components in blue, the regression components in red, and the common components in black, clearly highlighting the composite nature of the model.

$$\begin{aligned} &\min_{\beta,\beta_0,\xi,\xi',\xi^*} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}\xi_i + \mu\sum_{i=1}^{n}(\xi_i' + \xi_i^*) \\ &\text{s.t.} \begin{cases} g(X_i) - g(X_{j(i)}) \geq T_i - T_{j(i)} - \xi_i \\ \Delta_i(g(X_i) - T_i) \leq \xi_i^* \\ T_i - g(X_i) \leq \xi_i' \\ \xi_i, \xi_i', \xi_i^* \geq 0, \quad \forall i = 1, ..., n \end{cases} \end{aligned} \tag{3.5.14}$$

and predictions are made with,

$$\hat{g}(X^*) = \sum_{i=1}^{n} \alpha_i(K(X_i, X^*) - K(X_{j(i)}, X^*)) + \alpha_i^* K(X_i, X^*) - \Delta_i \alpha_i' K(X_i, X^*) + \beta_0 \tag{3.5.15}$$

The regularizer hyper-parameters C and μ now have a clear interpretation. C is the penalty associated with the regression method and μ is the penalty associated with the ranking method. By always fitting the hybrid models and tuning these two parameters, there is never a requirement to separately fit the regression or ranking methods as these would be automatically identified as superior in the tuning procedure. Moreover, the hybrid model retains the interpretability of the regression method and predictions can be interpreted as survival times. The hybrid method is implemented in **survivalsvm** as 'hybrid'. By Van Belle's own simulation studies, these models do not outperform the Cox PH with respect to Harrell's C.

SSVR-MRL Not all SSVMs can be considered a variant of the SSVM-Hybrid, though all prominent and commonly utilised suggestions do seem to have this formulation. One other algorithm of note is termed here the 'SSVM-MRL' [102, 103], which is a regression SSVM. The algorithm is identical to SVCR with one additional constraint.

SSVR-MRL

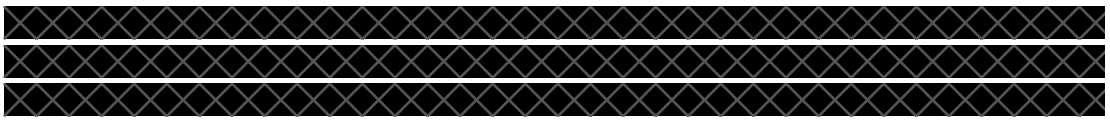
$$\min_{\beta,\beta_0,\xi,\xi^*,\xi'} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) + C^*\sum_{i=1}^{n}\xi_i'$$

$$\text{s.t.} \begin{cases} T_i - g(X_i) \leq \xi_i \\ \Delta_i(g(X_i) - T_i) \leq \xi_i^* \\ (1-\Delta_i)(g(X_i) - T_i - MRL(T_i|\hat{S})) \leq \xi_i' \\ \xi_i, \xi_i^*, \xi_i' \geq 0 \\ \forall i = 1, ..., n \end{cases} \tag{3.5.16}$$

where $MRL(T_i|\hat{S})$ is the 'mean residual lifetime' function [164]

$$MRL(\tau|\hat{S}) = \frac{\int_\tau^\infty \hat{S}(u)du}{\hat{S}(\tau)} \tag{3.5.17}$$

which is the area under the estimated survival curve (say by Kaplan Meier), $\hat{S}$, from point τ, weighted by the probability of being alive at point τ. This is interpreted as the expected remaining lifetime from point τ. On setting $C^* = 0$ and removing associated constraint three, this reduces exactly to the SVCR and similarly if there's no censoring then the standard regression setting is recovered. Unlike other strategies, no new hyper-parameters are introduced and Kaplan-Meier estimation should not noticeably impact run-time. There is no evidence of this model being used in practice, nor of any off-shelf implementation. Theoretically, the hybrid model could be expanded to include this extra penalty term and constraint (discussed below).



3.6. Neural Networks

Before starting the survey on neural networks, first a comment about their transparency and accessibility. Neural networks are infamously difficult to interpret and train, with some calling building and training neural networks an 'art' [118]. As discussed in the introduction of this thesis, whilst neural networks are not transparent with respect to their predictions, they are transparent with respect to implementation. In fact the simplest form of neural network, as seen below, is no more complex than a simple linear model. With regard to accessibility, whilst it is true that defining a custom neural network architecture is complex and highly subjective, established models are implemented with a default architecture and are therefore accessible 'off-shelf'.

3.6.1. Neural Networks for Regression

(Artificial) Neural networks (ANNs) are a class of model that fall within the greater paradigm of *deep learning*. The simplest form of ANN, a feed-forward single-hidden-layer network, is a relatively simple algorithm that relies on linear models, basic activation functions, and simple derivatives. A short introduction to feed-forward regression ANNs is provided to motivate the survival models. This focuses on single-hidden-layer models and increasing this to multiple hidden layers follows relatively simply.

The single hidden-layer network is defined through three equations

$$Z_m = \sigma(\alpha_{0m} + \alpha_m^T X_i), \quad m = 1, ..., M \tag{3.6.1}$$

$$T = \beta_{0k} + \beta_k^T Z, \quad k = 1, .., K \tag{3.6.2}$$

$$g_k(X_i) = \phi_k(T) \tag{3.6.3}$$

where $(X_1, ..., X_n) \overset{i.i.d.}{\sim} X$ are the usual training data, α_{0m}, β_0 are bias parameters, and $\theta = \{\alpha_m, \beta\}$ $(m = 1, .., , M)$ are model weights where M is the number of hidden units. K is the number of classes in the output, which for regression is usually $K = 1$. The function ϕ is a 'link' or 'activation function', which transforms the predictions in order to provide an outcome of the correct return type; usually in regression, $\phi(x) = x$. σ is the 'activation function', which transforms outputs from each layer. The α_m parameters are often referred to as 'activations'. Different activation functions may be used in each layer or the same used throughout, the choice is down to expert knowledge. Common activation functions seen in this section include the sigmoid function,

$$\sigma(v) = (1 + \exp(-v))^{-1} \tag{3.6.4}$$

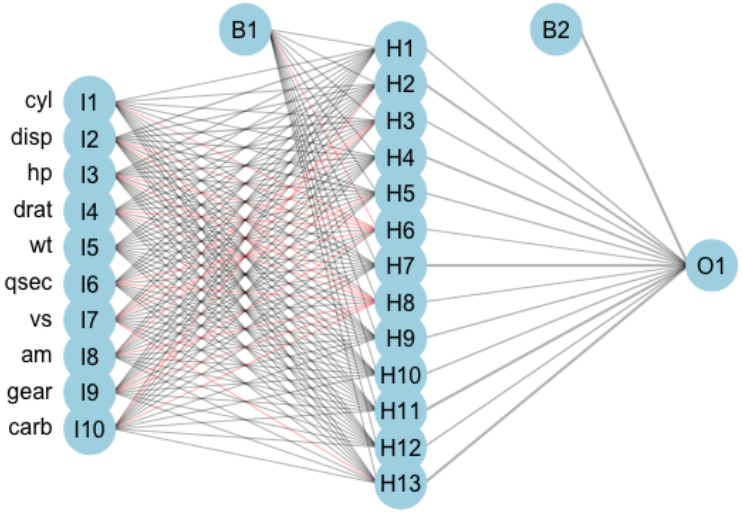


Figure 7: Single-hidden-layer artificial neural network with 13 hidden units fit on the `mtcars` [122] dataset using the **nnet** [222] package, and **gamlss.add** [286] for plotting. Left column are input variables, I1-I10, second column are 13 hidden units, H1-H13, right column is single output variable, O1. B1 and B2 are bias parameters.

tanh function,

$$\sigma(v) = \frac{\exp(v) - \exp(-v)}{\exp(v) + \exp(-v)} \tag{3.6.5}$$

and ReLU [224]

$$\sigma(v) = \max(0, v) \tag{3.6.6}$$

A single-hidden-layer model can also be expressed in a single equation, which highlights the relative simplicity of what may appear a complex algorithm.

$$g_k(X_i) = \sigma_0(\beta_{k0} + \sum_{h=1}^{H}(\beta_{kh}\sigma_h(\beta_{h0} + \sum_{m=1}^{M}\beta_{hm}X_{i;m})) \tag{3.6.7}$$

where H are the number of hidden units, β are the model weights, σ_h is the activation function in unit h, also σ_0 is the output unit activation, and $X_{i;m}$ is the ith observation features in the mth hidden unit.

An example feed-forward single-hidden-layer regression ANN is displayed in fig. 7. This model has 10 input units, 13 hidden units, and one output unit; two bias parameters are fit. The model is described as 'feed-forward' as there are no cycles in the node and information is passed forward from the input nodes (left) to the output node (right).

Back-Propagation The model weights, θ, in this section are commonly fit by 'back-propagation' although this method is often considered inefficient compared to more recent advances. A brief pseudo-algorithm for the process is provided below.

Let L be a chosen loss function for model fitting, let $\theta = (\alpha, \beta)$ be model weights, and let $J \in \mathbb{N}_{>0}$ be the number of iterations to train the model over. Then the back-propagation method is given by,

1. **For** $j = 1, ..., J$:

   Forward Pass

   i. Fix weights $\theta^{(j-1)}$.

   ii. Compute predictions $\hat{Y} := \hat{g}_k^{(j)}(X_i|\theta^{(j-1)})$ with eq. (3.6.7).

   Backward Pass

   iii. Calculate the gradients of the loss $L(\hat{Y}|\mathcal{D}_0)$.

   Update

   iv. Update $\alpha^{(r)}, \beta^{(r)}$ with gradient descent.

2. **End For**

In regression, a common choice for L is the squared loss,

$$L(\hat{g}, \theta|\mathcal{D}_0) = \sum_{i=1}^{n}(Y_i - \hat{g}(X_i|\theta))^2 \tag{3.6.8}$$

which may help illustrate how the training outcome, $(Y_1, ..., Y_n) \overset{i.i.d.}{\sim} Y$, is utilised for model fitting.

Making Predictions Once the model is fitted, predictions for new data follow by passing the testing data as inputs to the model with fitted weights,

$$g_k(X^*) = \sigma_0(\hat{\beta}_{k0} + \sum_{h=1}^{H}(\hat{\beta}_{kh}\sigma_h(\hat{\beta}_{h0} + \sum_{m=1}^{M}\hat{\beta}_{hm}X_m^*)) \tag{3.6.9}$$

Hyper-Parameters In practice, a regularization parameter, λ, is usually added to the loss function in order to help avoid overfitting. This parameter has the effect of shrinking model weights towards zero and hence in the context of ANNs regularization is usually referred to as 'weight decay'. The value of λ is one of three important hyper-parameters in all ANNs, the other two are: the range of values to simulate initial weights from, and the number of hidden units, M.

The range of values for initial weights is usually not tuned but instead a consistent range is specified and the neural network is trained multiple times to account for randomness in initialization.

The regularization parameter and number of hidden units, M, depend on each other and have a similar relationship to the learning rate and number of iterations in the GBMs (section 3.4). Like the GBMs, it is simplest to set a high number of hidden units and then tune the regularization parameter [23, 118]. Determining how many hidden layers to include, and how to connect them, is informed by expert knowledge and well beyond the scope of this thesis; decades of research has been required to derive sensible new configurations.

Training Batches ANNs can either be trained using complete data, in batches, or online. This decision is usually data-driven and will affect the maximum number of iterations used to train the algorithm; as such this will also often be chosen by expert-knowledge and not empirical methods such as cross-validation.

Neural Terminology Neural network terminology often reflects the structures of the brain. Therefore ANN units are referred to as nodes or neurons and sometimes the connections between neurons are referred to as synapses. Neurons are said to be 'fired' if they are 'activated'. The simplest example of activating a neuron is with the Heaviside activation function with a threshold of 0: $\sigma(v) = \mathbb{I}(v \geq 0)$. Then a node is activated and passes its output to the next layer if its value is positive, otherwise it contributes no value to the next layer.

3.6.2. Neural Networks for Survival Analysis

Surveying neural networks is a non-trivial task as there has been a long history in machine learning of publishing very specific data-driven neural networks with limited applications; this is also true in survival analysis. This does mean however that where limited developments for survival were made in other machine learning classes, ANN survival adaptations have been around for several decades. A review in 2000 by Schwarzer *et al.* surveyed 43 ANNs for diagnosis and prognosis published in the first half of the 90s, however only up to ten of these are specifically for survival data.[1] Of those, Schwarzer *et al.* deemed three to be 'naïve applications to survival data', and recommended for future research models developed by Liestøl *et al.* (1994) [197], Faraggi and Simon (1995) [78], and Biganzoli *et al.* (1998) [18].

This survey will not be as comprehensive as the 2000 survey, and nor has any survey since, although there have been several ANN reviews [251, 135, 232, 328, 334]. ANNs are considered to be a black-box model, with interpretability decreasing steeply as the number of hidden layers and nodes increases. In terms of accessibility there have been relatively few open-source packages developed for survival ANNs; where these are available the focus has historically been in Python, with no R implementations. The new **survivalmodels** [275] package,[2] implements these Python models via **reticulate** [301]. No recurrent neural netwoks are included in this survey though the survival models SRN [230] and RNN-Surv [99] are acknowledged.

This survey is made slightly more difficult as neural networks are often proposed for many different tasks, which are not necessarily clearly advertised in a paper's title or abstract. For example, many papers claim to use neural networks for survival analysis and make comparisons to Cox models, whereas the task tends to be death at a particular (usually 5-year) time-point (classification) [114, 203, 251, 252, 271], which is often not made clear until mid-way through the paper. Reviews and surveys have also conflated these different tasks, for example a very recent review concluded superior performance of ANNs over Cox models, when in fact this is only in classification [134] (section 5.3.3 (RM2)). To clarify, this form of classification task does fall into the general *field* of survival analysis, but not the survival *task* (box 3). Therefore this is not a comment on the classification task but a reason for omitting these models from this survey.

[1] Schwarzer conflates the prognosis and survival task, therefore it is not clear if all 10 of these are for time-to-event data (at least five definitely are).

[2] Created in order to run the experiments in chapter 7.

Using ANNs for feature selection (often in gene expression data) and computer vision is also very common in survival analysis, and indeed it is in this area that most success has been seen [10, 46, 61, 185, 213, 250, 270, 331, 335], but these are again beyond the scope of this survey.

The key difference between neural networks is in their output layer, required data transformations, the model prediction, and the loss function used to fit the model. Therefore the following are discussed for each of the surveyed models: the loss function for training, L, the model prediction type, $\hat{g}$, and any required data transformation. Notation is continued from the previous surveys with the addition of θ denoting model weights (which will be different for each model).

3.6.2.1. Probabilistic Survival Models

Unlike other classes of machine learning models, the focus in ANNs has been on probabilistic models. The vast majority make these predictions via reduction to binary classification (section 5.5.7.6). Whilst almost all of these networks implicitly reduce the problem to classification, most are not transparent in exactly how they do so and none provide clear or detailed interface points in implementation allowing for control over this reduction. Most importantly, the majority of these models do not detail how valid survival predictions are derived from the binary setting,[1] which is not just a theoretical problem as some implementations, such as the Logistic-Hazard model in **pycox** [172], have been observed to make survival predictions outside the range $[0, 1]$. This is not a statement about the performance of models in this section but a remark about the lack of transparency across all probabilistic ANNs.

Many of these algorithms use an approach that formulate the Cox PH as a non-linear model and minimise the partial likelihood. These are referred to as 'neural-Cox' models and the earliest appears to have been developed by Faraggi and Simon [78]. All these models are technically composites that first predict a ranking, however they assume a PH form and in implementation they all appear to return a probabilistic prediction.

ANN-COX
Faraggi and Simon [78] proposed a non-linear PH model

$$h(\tau|X_i, \theta) = h_0(\tau) \exp(\phi(X_i\beta)) \tag{3.6.10}$$

where ϕ is the sigmoid function and $\theta = \{\beta\}$ are model weights. This model, 'ANN-COX', estimates the prediction functional, $\hat{g}(X^*) = \phi(X^*\hat{\beta})$. The model is trained with the partial-likelihood function

$$L(\hat{g}, \theta|\mathcal{D}_0) = \prod_{i=1}^{n} \frac{\exp(\sum_{m=1}^{M} \alpha_m \hat{g}_m(X^*))}{\sum_{j \in \mathcal{R}_{t_i}} \exp(\sum_{m=1}^{M} \alpha_m \hat{g}_m(X^*))} \tag{3.6.11}$$

[1] One could assume they use procedures such as those described in Tutz and Schmid (2016) [298] but there is rarely transparent writing to confirm this.

where $\mathcal{R}_{t_i}$ is the risk group alive at t_i; M is the number of hidden units; $\hat{g}_m(X^*) = (1 + \exp(-X^*\hat{\beta}_m))^{-1}$; and $\theta = \{\beta, \alpha\}$ are model weights.

The authors proposed a single hidden layer network, trained using back-propagation and weight optimisation with Newton-Raphson. This architecture did not outerperform a Cox PH [78]. Further adjustments including (now standard) pre-processing and hyper-parameter tuning did not improve the model performance [208]. Further independent studies demonstrated worse performance than the Cox model [78, 327].

COX-NNET

COX-NNET [49] updates the ANN-COX by instead maximising the regularized partial log-likelihood

$$L(\hat{g}, \theta|\mathcal{D}_0, \lambda) = \sum_{i=1}^{n} \Delta_i \Big[\hat{g}(X_i) \; - \; \log\Big(\sum_{j \in \mathcal{R}_{t_i}} \exp(\hat{g}(X_j))\Big)\Big] + \lambda(\|\beta\|_2 + \|w\|_2) \tag{3.6.12}$$

with weights $\theta = (\beta, w)$ and where $\hat{g}(X_i) = \sigma(wX_i + b)^T\beta$ for bias term b, and activation function σ; σ is chosen to be the tanh function (3.6.5). In addition to weight decay, dropout [285] is employed to prevent overfitting. Dropout can be thought of as a similar concept to the variable selection in random forests, as each node is randomly deactivated with probability p, where p is a hyper-parameter to be tuned.

Independent simulation studies suggest that COX-NNET does not outperform the Cox PH [96].

DeepSurv

DeepSurv [156] extends these models to deep learning with multiple hidden layers. The chosen error function is the average negative log-partial-likelihood with weight decay

$$L(\hat{g}, \theta|\mathcal{D}_0, \lambda) = -\frac{1}{n^*} \sum_{i=1}^{n} \Delta_i \Big[\Big(\hat{g}(X_i) - \log \sum_{j \in \mathcal{R}_{t_i})} \exp(\hat{g}(X_j)\Big)\Big] + \lambda\|\theta\|_2^2 \tag{3.6.13}$$

where $n^* := \sum_{i=1}^{n} \mathbb{I}(\Delta_i = 1)$ is the number of uncensored observations and $\hat{g}(X_i) = \phi(X_i|\theta)$ is the same prediction object as the ANN-COX. State-of-the-art methods are used for data pre-processing and model training. The model architecture uses a combination of fully-connected and dropout layers. Benchmark experiments by the authors indicate that DeepSurv can outperform the Cox PH in ranking tasks [155, 156] although independent experiments do not confirm this [332].

Cox-Time

Kvamme *et al.* [174] build on these models by allowing time-varying effects. The loss function to minimise, with regularization, is given by

$$L(\hat{g}, \theta | \mathcal{D}_0, \lambda) = \frac{1}{n} \sum_{i:\Delta_i=1} \log \Big(\sum_{j \in \mathcal{R}_{t_i}} \exp[\hat{g}(X_j, T_i) - \hat{g}(X_i, T_i)] \Big) + \lambda \sum_{i:\Delta_i=1} \sum_{j \in \mathcal{R}_{t_i}} |\hat{g}(X_j, T_i)| \tag{3.6.14}$$

where $\hat{g} = \hat{g}_1, ..., \hat{g}_n$ is the same non-linear predictor but with a time interaction and λ is the regularization parameter. The model is trained with stochastic gradient descent and the risk set, $\mathcal{R}_{t_i}$, in the equation above is instead reduced to batches, as opposed to the complete dataset. ReLU activations [224] and dropout are employed in training. Benchmark experiments indicate good performance of Cox-Time, though no formal statistical comparisons are provided and hence no comment about general performance can be made.

ANN-CDP

One of the earliest ANNs that was noted by Schwarzer *et al.* [268] was developed by Liestøl *et al.* [197] and predicts conditional death probabilities (hence 'ANN-CDP'). The model first partitions the continuous survival times into disjoint intervals $\mathcal{I}_k, k = 1, ..., m$ such that $\mathcal{I}_k$ is the interval $(t_{k-1}, t_k]$. The model then studies the logistic Cox model (proportional odds) [59] given by

$$\frac{p_k(\mathbf{x})}{q_k(\mathbf{x})} = \exp(\eta + \theta_k) \tag{3.6.15}$$

where $p_k = 1 - q_k$, $\theta_k = \log(p_k(0)/q_k(0))$ for some baseline probability of survival, $q_k(0)$, to be estimated; η is the usual linear predictor, and $q_k = P(T \geq T_k | T \geq T_{k-1})$ is the conditional survival probability at time T_k given survival at time T_{k-1} for $k = 1, ..., K$ total time intervals. A logistic activation function is used to predict $\hat{g}(X^*) = \phi(\eta + \theta_k)$, which provides an estimate for $\hat{p}_k$.

The model is trained on discrete censoring indicators D_{ki} such that $D_{ki} = 1$ if individual i dies in interval $\mathcal{I}_k$ and 0 otherwise. Then with K output nodes and maximum likelihood estimation to find the model parameters, $\hat{\eta}$, the final prediction provides an estimate for the conditional death probabilities $\hat{p}_k$. The negative log-likelihood to optimise is given by

$$L(\hat{g}, \theta | \mathcal{D}_0) = \sum_{i=1}^{n} \sum_{k=1}^{m_i} [D_{ki} \log(\hat{p}_k(X_i)) + (1 - D_{ki}) \log(\hat{q}_k(X_i))] \tag{3.6.16}$$

where m_i is the number of intervals in which observation i is not censored.

Liestøl *et al.* discuss different weighting options and how they correspond to the PH assumption. In the most generalised case, a weight-decay type regularization is applied to the model weights given by

$$\alpha \sum_l \sum_k (w_{kl} - w_{k-1,l})^2 \tag{3.6.17}$$

where w are weights, and α is a hyper-parameter to be tuned, which can be used

alongside standard weight decay. This corresponds to penalizing deviations from proportionality thus creating a model with approximate proportionality. The authors also suggest the possibility of fixing the weights to be equal in some nodes and different in others; equal weights strictly enforces the proportionality assumption. Their simulations found that removing the proportionality assumption completely, or strictly enforcing it, gave inferior results. Comparing their model to a standard Cox PH resulted in a 'better' negative log-likelihood, however this is not a precise evaluation metric and an independent simulation would be preferred. Finally Listøl *et al.* included a warning "The flexibility is, however, obtained at unquestionable costs: many parameters, difficult interpretation of the parameters and a slow numerical procedure" [197].

PLANN

Biganzoli *et al.* (1998) [18] studied the same proportional-odds model as the ANN-CDP [197]. Their model utilises partial logistic regression [75] with added hidden nodes, hence 'PLANN'. Unlike ANN-CDP, PLANN predicts a smoothed hazard function by using smoothing splines. The continuous time outcome is again discretised into disjoint intervals $t_m, m = 1, ..., M$. At each time-interval, t_m, the number of events, d_m, and number of subjects at risk, n_m, can be used to calculate the discrete hazard function,[1]

$$\hat{h}_m = \frac{d_m}{n_m}, m = 1, ..., M \tag{3.6.18}$$

This quantity is used as the target to train the neural network. The survival function is then estimated by the Kaplan-Meier type estimator,

$$\hat{S}(\tau) = \prod_{m:t_m \leq \tau} (1 - \hat{h}_m) \tag{3.6.19}$$

The model is fit by employing one of the more 'usual' survival reduction strategies in which an observation's survival time is treated as a covariate in the model [298]. As this model uses discrete time, the survival time is discretised into one of the M intervals. This approach removes the proportional odds constraint as interaction effects between time and covariates can be modelled (as time-updated covariates). Again the model makes predictions at a given time m, $\phi(\theta_m + \eta)$, where η is the usual linear predictor, θ is the baseline proportional odds hazard $\theta_m = \log(h_m(0)/(1 - h_m(0))$. The logistic activation provides estimates for the discrete hazard,

$$h_m(X_i) = \frac{\exp(\theta_m + \hat{\eta})}{1 + \exp(\theta_m + \hat{\eta})} \tag{3.6.20}$$

which is smoothed with cubic splines [75] that require tuning.

[1] Derivation of this as a 'hazard' estimator follows trivially by comparison to the Nelson-Aalen estimator.

A cross-entropy error function is used for training

$$L(\hat{h}, \theta|\mathcal{D}_0, a) = -\sum_{m=1}^{M} \Big[\hat{h}_m \log\Big(\frac{h_l(X_i, a_l)}{\hat{h}_m}\Big) + (1-\hat{h}_m)\log\Big(\frac{1-h_l(X_i, a_l)}{1-\hat{h}_m}\Big)\Big] n_m \tag{3.6.21}$$

where $h_l(X_i, a_l)$ is the discrete hazard h_l with smoothing at mid-points a_l. Weight decay can be applied and the authors suggest $\lambda \approx 0.01 - 0.1$ [18], though they make use of an AIC type criterion instead of cross-validation.

This model makes smoothed hazard predictions at a given time-point, τ, by including τ in the input covariates X_i. Therefore the model first requires transformation of the input data by replicating all observations and replacing the single survival indicator Δ_i, with a time-dependent indicator D_{ik}, the same approach as in ANN-CDP. Further developments have extended the PLANN to Bayesian modelling, and for competing risks [17].

No formal comparison is made to simpler model classes. The authors recommend ANNs primarily for exploration, feature selection, and understanding underlying patterns in the data [17].

Nnet-survival

Aspects of the PLANN algorithm have been generalised into discrete-time survival algorithms in several papers [96, 173, 206, 288]. Various estimates have been derived for transforming the input data to a discrete hazard or survival function. Though only one is considered here as it is the most modern and has a natural interpretation as the 'usual' Kaplan-Meier estimator for the survival function. Others by Street (1998) [288] and Mani (1999) [206] are acknowledged. The discrete hazard estimator (eq. (3.6.18)), $\hat{h}$, is estimated and these values are used as the targets for the ANN. For the error function, the mean negative log-likelihood for discrete time [173] is minimised to estimate $\hat{h}$,

$$\begin{aligned} L(\hat{h}, \theta|\mathcal{D}_0) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k(T_i)} (&\mathbb{I}(T_i = \tau_j, \Delta_i = 1)\log[\hat{h}_i(\tau_j)] + \\ &(1 - \mathbb{I}(T_i = \tau_j, \Delta_i = 1))\log(1 - \hat{h}_i(\tau_j))) \end{aligned} \tag{3.6.22}$$

where $k(T_i)$ is the time-interval index in which observation i dies/is censored, τ_j is the jth discrete time-interval, and the prediction of $\hat{h}$ is obtained via

$$\hat{h}(\tau_j|\mathcal{D}_0) = [1 + \exp(-\hat{g}_j(\mathcal{D}_0))]^{-1} \tag{3.6.23}$$

where $\hat{g}_j$ is the jth output for $j = 1, ..., m$ discrete time intervals. The number of units in the output layer for these models corresponds to the number of discrete-time intervals. Deciding the width of the time-intervals is an additional hyper-parameter to consider.

Gensheimer and Narasimhan's 'Nnet-survival' [96] has two different implementations. The first assumes a PH form and predicts the linear predictor in the final layer, which can then be composed to a distribution. Their second 'flexible' approach instead predicts the log-odds of survival in each node, which are then

converted to a conditional probability of survival, $1-h_j$, in a given interval using the sigmoid activation function. The full survival function can be derived with eq. (3.6.19). The model has been demonstrated not to outperform the Cox PH w.r.t. Harrell's C or the Graf (Brier) score [96].

PC-Hazard

Kvamme and Borgan deviate from nnet-survival in their 'PC-Hazard' [173] by first considering a discrete-time approach with a softmax activation function influenced by multi-class classification. They expand upon this by studying a piecewise constant hazard function in continuous time and defining the mean negative log-likelihood as

$$L(\hat{g},\theta|\mathcal{D}_0) = -\frac{1}{n}\sum_{i=1}^{n}\Big(\Delta_i X_i \log\tilde{\eta}_{k(T_i)} - X_i\tilde{\eta}_{k(T_i)}\rho(T_i) - \sum_{j=1}^{k(T_i)-1}\tilde{\eta}_j X_i\Big) \tag{3.6.24}$$

where $k(T_i)$ and τ_i is the same as defined above, $\rho(t) = \frac{t-\tau_{k(t)-1}}{\Delta\tau_{k(t)}}$, $\Delta\tau_j = \tau_j - \tau_{j-1}$, and $\tilde{\eta}_j := \log(1+\exp(\hat{g}_j(X_i))$ where again $\hat{g}_j$ is the jth output for $j = 1, ..., m$ discrete time intervals. Once the weights have been estimated, the predicted survival function is given by

$$\hat{S}(\tau, X^*|\mathcal{D}_0) = \exp(-X^*\tilde{\eta}_{k(\tau)}\rho(\tau))\prod_{j=1}^{k(\tau)-1}\exp(-\tilde{\eta}_j(X^*)) \tag{3.6.25}$$

Benchmark experiments indicate similar performance to nnet-survival [173], an unsurprising result given their implementations are identical with the exception of the loss function [173], which is also similar for both models. A key result found that varying values for interval width lead to significant differences and therefore should be carefully tuned.

DNNSurv

A very recent (pre-print) approach [332] instead first computes 'pseudo-survival probabilities' and uses these to train a regression ANN with sigmoid activation and squared error loss. These pseudo-probabilities are computed using a jackknife-style estimator given by

$$\tilde{S}_{ij}(T_{j+1}, \mathcal{R}_{t_j}) = n_j\hat{S}(T_{j+1}|\mathcal{R}_{t_j}) - (n_j-1)\hat{S}^{-i}(T_{j+1}|\mathcal{R}_{t_j}) \tag{3.6.26}$$

where $\hat{S}$ is the IPCW weighted Kaplan-Meier estimator (defined below) for risk set $\mathcal{R}_{t_j}$, $\hat{S}^{-i}$ is the Kaplan-Meier estimator for all observations in $\mathcal{R}_{t_j}$ excluding observation i, and $n_j := |\mathcal{R}_{t_j}|$. The IPCW weighted KM estimate is found via the IPCW Nelson-Aalen estimator,

$$\hat{H}(\tau|\mathcal{D}_0) = \sum_{i=1}^{n}\int_0^\tau \frac{\mathbb{I}(T_i \le u, \Delta_i = 1)\hat{W}_i(u)}{\sum_{j=1}^{n}\mathbb{I}(T_j \ge u)\hat{W}_j(u)}\,du \tag{3.6.27}$$

where $\hat{W}_i, \hat{W}_j$ are subject specific IPC weights.

In their simulation studies, they found no improvement over other proposed neural networks. Arguably the most interesting outcome of their paper are comparisons of multiple survival ANNs at specific time-points, evaluated with C-index and Brier score. Their results indicate identical performance from all models. They also provide further evidence of neural networks not outperforming a Cox PH when the PH assumption is valid. However, in their non-PH dataset, DNNSurv appears to outperform the Cox model (no formal tests are provided). Data is replicated similarly to previous models except that no special indicator separates censoring and death, this is assumed to be handled by the IPCW pseudo probabilities.

DeepHit
DeepHit [191] was originally built to accommodate competing risks, but only the non-competing case is discussed here [174]. The model builds on previous approaches by discretising the continuous time outcome, and makes use of a composite loss. It has the advantage of making no parametric assumptions and directly predicts the probability of failure in each time-interval (which again correspond to different terminal nodes), i.e. $\hat{g}(\tau_k|\mathcal{D}_1) = \hat{P}(T^* = \tau_k|X^*)$ where again $\tau_k, k = 1, ..., K$ are the distinct time intervals. The estimated survival function is found with $\hat{S}(\tau_K|X^*) = 1 - \sum_{k=1}^{K} \hat{g}_i(\tau_k|X^*)$. ReLU activations were used in all fully connected layers and a softmax activation in the final layer. The losses in the composite error function are given by

$$L_1(\hat{g}, \theta|\mathcal{D}_0) = -\sum_{i=1}^{N}[\Delta_i \log(\hat{g}_i(T_i)) + (1 - \Delta_i) \log(\hat{S}_i(T_i))] \tag{3.6.28}$$

and

$$L_2(\hat{g}, \theta|\mathcal{D}_0, \sigma) = \sum_{i \neq j} \Delta_i \mathbb{I}(T_i < T_j)\sigma(\hat{S}_i(T_i), \hat{S}_j(T_i)) \tag{3.6.29}$$

for some convex loss function σ and where $\hat{g}_i(t) = \hat{g}(t|X_i)$. Again these can be seen to be a cross-entropy loss and a ranking loss. Benchmark experiments demonstrate the model outperforming the Cox PH and RSFs [191] with respect to separation, and an independent experiment supports these findings [174]. However, the same independent study demonstrated worse performance than a Cox PH w.r.t. the integrated Brier score [109].

3.6.2.2. Deterministic Survival Models

Whilst the vast majority of survival ANNs have focused on probabilistic predictions (often via ranking), a few have also tackled the deterministic or 'hybrid' problem.

RankDeepSurv
Jing *et al.* [148] observed the past two decades of research in survival ANNs and then published a completely novel solution, RankDeepSurv, which makes predictions for the survival time $\hat{T} = (\hat{T}_1, ..., \hat{T}_n)$. They proposed a composite loss

function

$$L(\hat{T}, \theta | \mathcal{D}_0, \alpha, \gamma, \lambda) = \alpha L_1(\hat{T}, T, \Delta) + \gamma L_2(\hat{T}, T, \Delta) + \lambda \|\theta\|_2^2 \tag{3.6.30}$$

where θ are the model weights, $\alpha, \gamma \in \mathbb{R}_{>0}$, λ is the shrinkage parameter, by a slight abuse of notation $T = (T_1, ..., T_n)$ and $\Delta = (\Delta_1, ..., \Delta_n)$, and

$$L_1(\hat{T}, \theta | \mathcal{D}_0) = \frac{1}{n} \sum_{\{i: I(i)=1\}} (\hat{T}_i - T_i)^2; \quad I(i) = \begin{cases} 1, & \Delta_i = 1 \cup (\Delta_i = 0 \cap \hat{T}_i \leq T_i) \\ 0, & \text{otherwise} \end{cases} \tag{3.6.31}$$

$$L_2(\hat{T}, \theta | \mathcal{D}_0) = \frac{1}{n} \sum_{\{i,j: I(i,j)=1\}}^{n} [(T_j - T_i) - (\hat{T}_j - \hat{T}_i)]^2; \quad I(i,j) = \begin{cases} 1, & T_j - T_i > \hat{T}_j - \hat{T}_i \\ 0, & \text{otherwise} \end{cases} \tag{3.6.32}$$

where $\hat{T}_i$ is the predicted survival time for observation i. A clear contrast can be made between these loss functions and the constraints used in SSVM-Hybrid [306] (section 3.5.2). L_1 is the squared second constraint in eq. (3.5.4) and L_2 is the squared first constraint in eq. (3.5.4). However L_1 in RankDeepSurv discards the squared error difference for all censored observations when the prediction is lower than the observed survival time; which is problematic as if someone is censored at time T_i then it is guaranteed that their true survival time is greater than T_i (this constraint may be more sensible if the inequality were reversed). An advantage to this loss is, like the SSVM-Hybrid, it enables a survival time interpretation for a ranking optimised model; however these 'survival times' should be interpreted with care.

The authors propose a model architecture with several fully connected layers with the ELU [54] activation function and a single dropout layer. Determining the success of this model is not straightforward. The authors claim superiority of RankDeepSurv over Cox PH, DeepSurv, and RSFs however this is an unclear comparison (section 5.3.3 (RM2)) that requires independent study.